AD-A021 224

MULTIDIMENSIONAL PREFERENTIAL STRATEGIES

John D. Matheson

Analytic Services, Incorporated

Prepared for:

Deputy Chief of Staff, Research and Development (Air Force)

November 1975

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: SDN 75-3	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): MULTIDIMENSIONAL PREFERENTIAL STRATEGIES		5. TYPE OF REPORT & PERIOD COVERED: Strategic Division Note
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): John D. Matheson		8. CONTRACT OR GRANT NUMBER(s): F44620-76-C-0010
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Analytic Services Inc. (ANSER) 5613 Leesburg Pike, Falls Church, VA 22041		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE: December 1975
		13. NUMBER OF PAGES: 162
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Directorate of Operational Requirements Hq USAF Wash., D.C. 20330		15. SECURITY CLASS. (of this report): UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited. It may be released to the National Technical Information Service for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Game Theory	Antimissile Defense
Operations Research	Optimization
Linear Programming	Computer Programming
Parametric Linear Programming	Mathematical Analysis
Strategic Warfare	Military Strategy

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents a method of solving weapon allocation games involving many weapon types and many target types. Numerical solutions are obtained by the PATH method, a form of parametric linear programming. Two computer programs are listed and explained, PATH87 for two-sided games and the simpler PATH87A for one-sided optimizations. Both are copiously illustrated by sample runs. Other applications of the programs are discussed in general terms.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (Continued)

The PATH method offers unique advantages of speed and flexibility in solving problems facing defense analysts, and it is hoped that publication of this report through the National Technical Information Service of the Defense Documentation Center will make this method more widely available. Also, the method has features which can be applied to many problems of resource allocation facing nondefense planners.

1

STRATEGIC DIVISION NOTE

SDN 75-3

MULTIDIMENSIONAL PREFERENTIAL STRATEGIES

November 1975

Prepared by
John D. Matheson

Approved by
J. A. Englund, Manager, Strategic Division

COPY A

ansér analytic services inc.

PREFACE

For over 8 years, the author has conducted a continuing research project that has produced new computer programs, some of which have wide applicability to allocation of resources (such as weapons) to objectives (such as targets). Some of the programs are general in nature and have been used by other ANSER analysts as well as the author to solve a variety of weapon systems analysis problems in our work for the Deputy Chief of Staff/Research and Development, Headquarters, United States Air Force. Several programs have also been given to various government agencies or defense contractors. Documentation of these has been minimal.

This report describes companion programs, PATH87 and PATH87A, that are very general in nature and which are milestones in a long evolutionary development. This report was originally drafted under the same title in March 1971, but the final version was not completed because of the press of other business. However, it was circulated in draft form to a number of defense contractors, some of whom were kind enough to list the draft as a reference in their own publications. The current edition represents a minor revision and update of the 1971 draft.

It has been demonstrated that the PATH method offers unique advantages of speed and flexibility in solving problems facing defense analysts and it is hoped that publication of this report through the National Technical Information Service of the Defense Documentation Center will make this method more widely available. Also, the method has features which can be applied to many problems of resource allocation facing nondefense planners.

TABLE OF CONTENTS

Preceding page blank

TABLE OF CONTENTS—Continued

TABLE OF CONTENTS—Continued

LIST OF FIGURES

MULTIDIMENSIONAL PREFERENTIAL STRATEGIES

I. INTRODUCTION

This report presents a method for solving a large class of resource allocation problems that are multidimensional in the sense of having many types of resources and many types of objects to which resources are allocated. The class of problems is restricted to those in which discrete quantities of resources are allocated to any object, for example, integral numbers of weapons to a target. The solution method involves parametric linear programming.

Historically, the method was developed in order to generalize earlier ANSER work (References 1 and 2). The essential features of Sections II and III were the subject of an oral seminar at the ORSA meeting, 1-3 May 1968, San Francisco, California. (Paper TP 1.12, A Generalized Weapon Allocation Game.)

A. The Problem

The general problem of the class with which we are dealing is formulated as a two-sided mathematical game in Section II. The formulation includes one-sided optimizations as special cases where one of the players of the game has zero resources.

An important feature of the formulation is the definition of an allocation, or strategy, in such a way that the solution variables appear linearly with constant coefficients in the objective function and the constraints. Any nonlinear features of the problem are incorporated in the computation of those coefficients.

Physical feasibility of solutions is ignored, and the solution strategies may contain fractional values that are apparently not feasible. However, these fractional strategies can be realized by one or more precisely equivalent, feasible, mixed strategies in nearly every two-sided game of practical interest. In most one-sided optimizations, the physical infeasibility, if any, is a local one of no great significance.

Solution strategies are obtained by solving simultaneous linear equations, with the matrix of coefficients for one player being the transpose of that for the other player. This common matrix constitutes a basis for the solution, called here a Q-basis.

B. Path Method

The use of parametric linear programming to determine the Q-basis is discussed in Section III.

Beginning with some set of resources, usually zero, for which the Q-basis is known, resources are then varied continuously in any arbitrarily prescribed manner; that is, along any prescribed path in the space of resources. The path method makes appropriate changes in the Q-basis at critical points of the path.

Solutions for the two players are alternated along the path, with a resource parameter being used for the player whose resources are changing and a marginal-value parameter for the other player.

C. Computer Programs

Two computer programs are included in the report. The PATH87 program, designed to solve multidimensional two-sided weapon-allocation games for point targets, is listed and discussed in Section IV. This program is the 1971 version in an evolutionary sequence pointed toward greater capacity, speed, flexibility, and reliability.

The other program is an abbreviated version designed to optimize one-sided weapon allocations for point targets. Called PATH87A, it is listed and discussed in Section V.

Readers who are not interested in the details of computer programming should skip Sections IV and V.

D. Preferential Strategies

Illustrative examples of actual runs of both programs are given in Section VI. One purpose of the section is to show someone who may have skipped the earlier sections how to use the programs, how to input data, and how to read the output. A second purpose is to point out and explain salient characteristics of typical solutions.

E. Other Applications

The path method applies to a variety of resource allocation problems that can be solved by suitable modifications of the two programs. Section VII lists some features of problems that have been solved and discusses the modifications in general terms.

II. THE PROBLEM

A multidimensional resource allocation problem of the class with which we are concerned can be formulated as a matrix game. In this section, we will give a general formulation of the game, including general solution conditions. We will then develop more detailed conditions in the form of a Q-basis, consisting of a certain coefficient matrix and some associated vectors. Finally, we will show that a solution can be computed directly from the Q-basis if certain conditions are satisfied.

A. Problem Formulation

1. Players, Resources, and Objects

In general terms, there are two opposing players, each having resources of different types to be allocated to specific objects (or activities) of which there are different types. One player is called the minimizing player because he seeks an allocation that will minimize some common measure of value. His opponent is called the maximizing player. Each player must make his own allocation in ignorance of his opponent's specific allocation, but he does know all the resources available to the other.

For illustration, an attacker would try to allocate a variety of weapons to a variety of targets in such a way as to minimize the expected value of the surviving targets, whereas a defender would try to allocate defensive weapons so as to maximize expected surviving value.

Let us adopt the following notation:

M = number of types of minimizing resources

N = number of types of maximizing resources

Preceding page blank

G = number of types of objects.

Let us also represent the number of units of each type by:

A_m for $m = 1, \cdots, M$

D_n for $n = 1, \cdots, N$

T_g for $g = 1, \cdots, G$.

Most of our subsequent analysis is concerned with the general case where both players have some resources. However, one should observe that the analysis is also valid if one of the players has no resources, in which case the problem becomes a one-sided optimization. As a convention, we will suppose this case to be represented by $N = 0$, so that we will always have $M \geq 1$ and $G \geq 1$.

2. Elementary Strategies

An elementary strategy is defined as any allocation by one player to a single object of a particular type. It consists of some number of units of each of that player's resource types. We will denote an elementary strategy for the minimizing player by the vector,

$$\alpha_i^g \equiv (\alpha_{i1}^g, \cdots, \alpha_{im}^g, \cdots, \alpha_{iM}^g)$$

and one for the maximizing player by the vector,

$$\delta_j^g \equiv (\delta_{1j}^g, \cdots, \delta_{nj}^g, \cdots, \delta_{Nj}^g) \quad .$$

The superscript g identifies the object type to which the elementary strategy applies. The subscript i (or j) identifies the particular elementary strategy among all those applying to object type g, under the assumption that they are arranged in some numerical order, the actual arrangement being immaterial. The subscript m (or n) distinguishes the resource type, so that the component α_{im}^g is the number of type m resources in the ith elementary minimizing strategy for object type g. We will call an elementary strategy with all zero components a null strategy.

3. General Strategies

We will now define a general strategy (or simply a strategy) for either player as a specific allocation of all his resources among all the objects. A strategy can be described in terms of elementary strategies by specifying the fraction of the number of objects of each type on which each elementary strategy is used.

For the minimizing player, we will denote by x_i^g the fraction of the number of objects of type g on which the elementary strategy α_i^g is used. The components, x_i^g, must satisfy three conditions: every component must be non-negative; every object must have some elementary strategy, if only the null strategy; and all resources must be used. These conditions can be represented by:

$$\begin{aligned} x_i^g &\geq 0 \text{ for every } g,i \\ \sum_i x_i^g &= 1 \text{ for every } g \qquad (1) \\ \sum_g \sum_i T_g \, \alpha_{im}^g \, x_i^g &= A_m \text{ for every } m \quad . \end{aligned}$$

If the components are arranged sequentially, they form a strategy vector:

$$X \equiv (x_1^1, x_2^1, \cdots, x_i^g, \cdots, x_i^G, \cdots) \quad ,$$

which defines an allocation of minimizing resources to objects.

In similar fashion, a strategy vector for the maximizing player is given by:

$$Y \equiv (y_1^1, y_2^1, \cdots, y_j^g, \cdots, y_j^G, \cdots) \quad ,$$

with conditions,

$$y_j^g \geq 0 \text{ for every } g,j$$

$$\sum_j y_j^g = 1 \text{ for every } g \qquad (2)$$

$$\sum_g \sum_j T_g \, \delta_{nj}^g \, y_j^g = D_n \text{ for every } n \quad .$$

At this point, we may observe that, in a one-sided optimization problem (characterized by N = 0), the only allowable elementary maximizing strategy on each object is the null strategy, $\delta_1^g = (0)$. With this restriction, $y_1^g = 1$, and the only maximizing strategy vector becomes

$$Y = (\overset{G \text{ times}}{1, \cdots, 1}) \quad .$$

4. Value Functions

The statement of a problem must provide some way for evaluating the worth of a strategy and making a choice of the "best" strategy. The basis of this evaluation is the value function, one of which must be specified for each object type. Each value function must define a value associated with every allowable combination of an elementary minimizing strategy and an elementary maximizing strategy on the same single object. Thus, if elementary strategies α_i^g and δ_j^g occur on the same object of type g, the value may be defined as v_{ij}^g. The value function for object type g then consists of the matrix,

$$V^g = (v_{ij}^g) \quad .$$

There must be a different value function for each object type; otherwise object types could be combined to reduce the dimensions of the problem. The several value functions must be expressed in terms of a common unit, or measure of value, but they need not have similar formulations. Indeed, an explicit formulation is not required at all, and a table of arbitrary values will serve the purpose.

There is great latitude for specifying value functions. For example, in a weapon allocation problem, an initial value might be specified for a single target of each type and a formula given for computing expected surviving value for any pair of elementary strategies. The value function in a transportation problem might be the cost matrix of mileages between sources and destinations.

Figure 1 illustrates the relationship between elements of a value function and associated strategies for a single object type. Figure 2 illustrates a useful concept of the combined value functions and strategies, where the blocks for each group symbolize sets of elements like those in Figure 1.

5. Objective Function

When the opposing players use specific general strategies, X and Y, each in ignorance of the other's strategy, there is an aggregate expected value which is a function of the strategies and is called the objective function.

In developing the objective function, let us consider first a single object of type g, and suppose α_i^g is the elementary minimizing strategy on this object. The probability that the elementary maximizing strategy, δ_j^g, is used on this object is y_j^g, i.e., the fraction of objects of this type on which the strategy δ_j^g is used. The expected value associated with this single object is then given by

$$\sum_j v_{ij}^g \, y_j^g \quad .$$

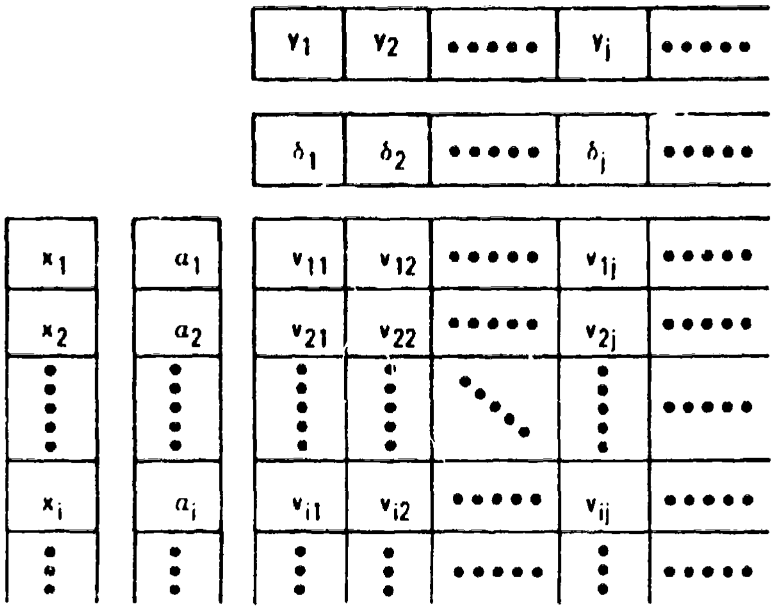


FIGURE 1
VALUE FUNCTION FOR A SINGLE OBJECT TYPE
(Omitting superscript g)

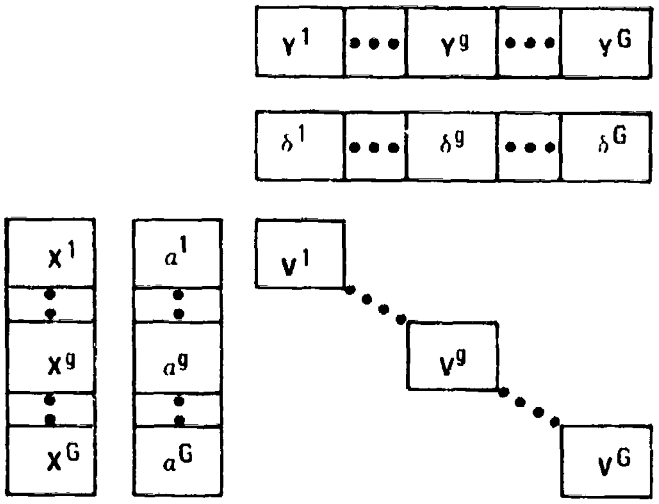


FIGURE 2
COMBINED VALUE FUNCTIONS

The number of objects on which the strategy α_i^g is used is $T_g\ x_i^g$. Hence, the sum of the expected values on all objects of type g is given by

$$\sum_i T_g\ x_i^g \sum_j v_{ij}^g\ y_j^g\ .$$

The sum of these values over all object types is the objective function, which can be written

$$V(X,Y) = \sum_g \sum_i \sum_j x_i^g\ (T_g\ v_{ij}^g)\ y_j^g\ .$$

This is the function that one player seeks to minimize by choosing a strategy X subject to the constraints (1). The other player seeks to maximize it by choosing a strategy Y subject to the constraints (2). The problem of finding best strategies for each player is a mathematical game.

In the foregoing formulation, the objective function is linear in the components of either the X or Y strategy, which permits linear programming methods to be used for the solution. A similar formulation for one-sided optimizations has been used by Dianich and Hennig (Reference 3).

6. Solution

For a solution, we adopt the standard criterion that a best strategy for the minimizing player is one that minimizes the maximum value his opponent can obtain against it. Conversely, a best strategy for the maximizing player is one that maximizes the minimum value his opponent can obtain against it. The theory of mathematical games guarantees that such strategies exist, although they need not be unique, and that the min-max value is equal to the max-min value, which is called the value of the game. If X* and Y* are used to denote solution strategies and V* the value of the game, then

$$V^* = \min_X \max_Y V(X,Y)$$

$$= \max_Y \min_X V(X,Y)$$

$$= V(X^*,Y^*) \quad .$$

Another way of expressing the minimax conditions is

$$V(X^*,Y) \le V^* \text{ for all } Y$$

$$V(X,Y^*) \ge V^* \text{ for all } X \quad ,$$

which is to say that, if either player uses a solution strategy, the other player cannot find a better strategy than his own solution strategy.

B. Equivalent Dual Linear Programs

Charnes (Reference 4) has shown that a constrained game is equivalent to dual linear programs in which the strategy vectors are augmented by components that are Lagrangian multipliers. In our case, the added components are denoted by superscript zeros or subscript zeros. The primal program can be written:

$$\text{Maximize} \quad \sum_n x_n^o D_n + \sum_g x_o^g T_g$$

$$\text{subject to:} \quad \sum_g \sum_i x_i^g (T_g \alpha_{im}^g) = A_m$$

$$\sum_i x_i^g = 1$$

$$\sum_n x_n^o \delta_{nj}^g + x_o^g + \sum_i x_i^g v_{ij}^g \le 0$$

$$x_i^g \ge 0 \quad ,$$

where all indexes are to be read as greater than zero unless explicitly stated as zero. Similarly, the dual program is:

$$\text{Minimize} \quad \sum_m A_m y_m^o + \sum_g T_g y_o^g \quad ,$$

$$\text{subject to:} \quad \sum_g \sum_j (T_g \delta_{nj}^g) y_j^g = D_n$$

$$\sum_j y_j^g = 1$$

$$\sum_m \alpha_{im}^g y_m^o + y_o^g + \sum_j v_{ij}^g y_j^g \geq 0$$

$$y_j^g \geq 0 \quad .$$

Charnes shows that solutions of the dual programs exist and that the value of the equivalent game is

$$V^* = -\sum_n x_n^o D_n - \sum_g x_o^g T_g$$

$$= -\sum_m A_m y_m^o - \sum_g T_g y_o^g \quad .$$

From this result, it is natural to interpret $-x_n^o$ and $-y_m^o$ as marginal values or values per unit of the respective resource types, and to interpret $-x_o^g$ and $-y_o^g$ as intercept values per object of the respective object types.

C. The Q-Basis

Balinski and Tucker (Reference 5) display a scheme for a Charnes-type formulation. We suppose such a scheme to exist for our problem. We then suppose that a solution is known, X* and Y*, and we delete from the scheme all those rows and columns for which solution components x_i^g and y_j^g are zero. The remaining elements, arranged to fit the block-diagonal structure of the problem, are illustrated in Figures 3 and 4 as what we call a Q-basis. It is a

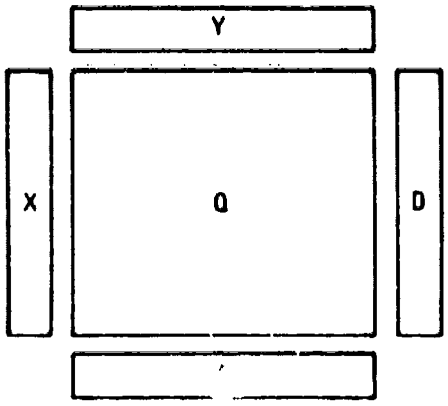


FIGURE 3
MATRIX AND VECTORS IN Q-BASIS

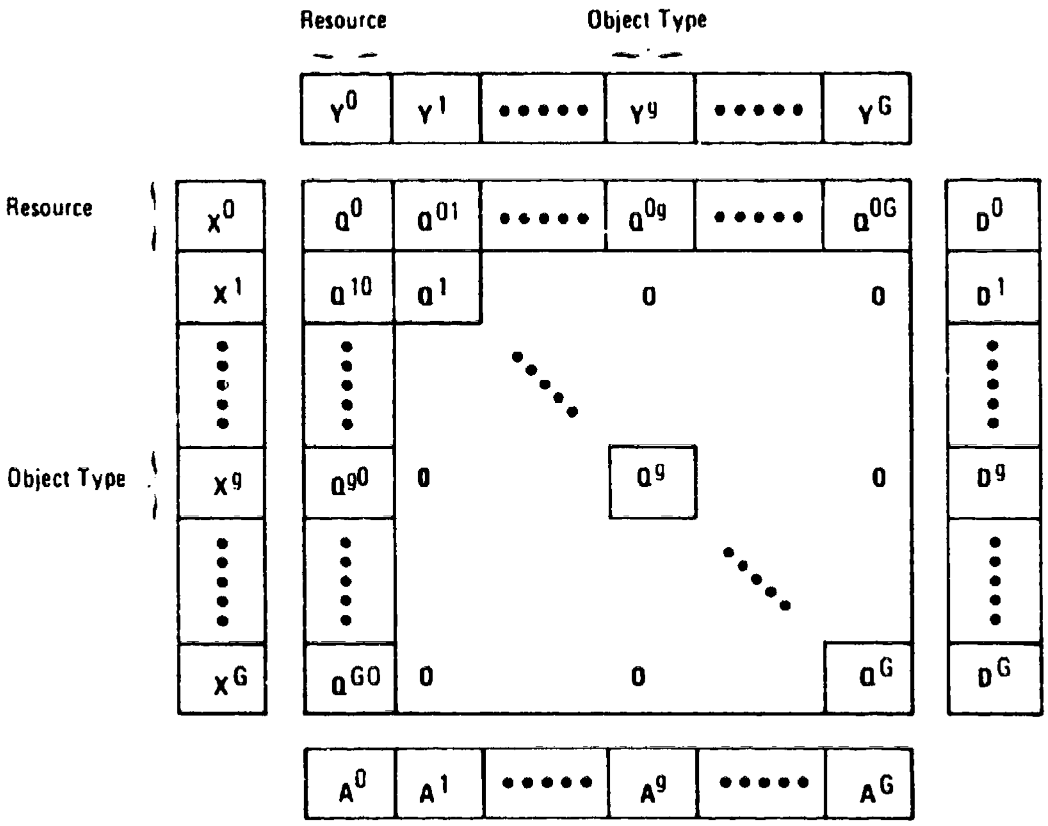


FIGURE 4
SCHEMATIC BLOCK PARTITIONING IN Q-BASIS

representation of the equations satisfied by the dual solutions, considering the inequalities as side conditions.

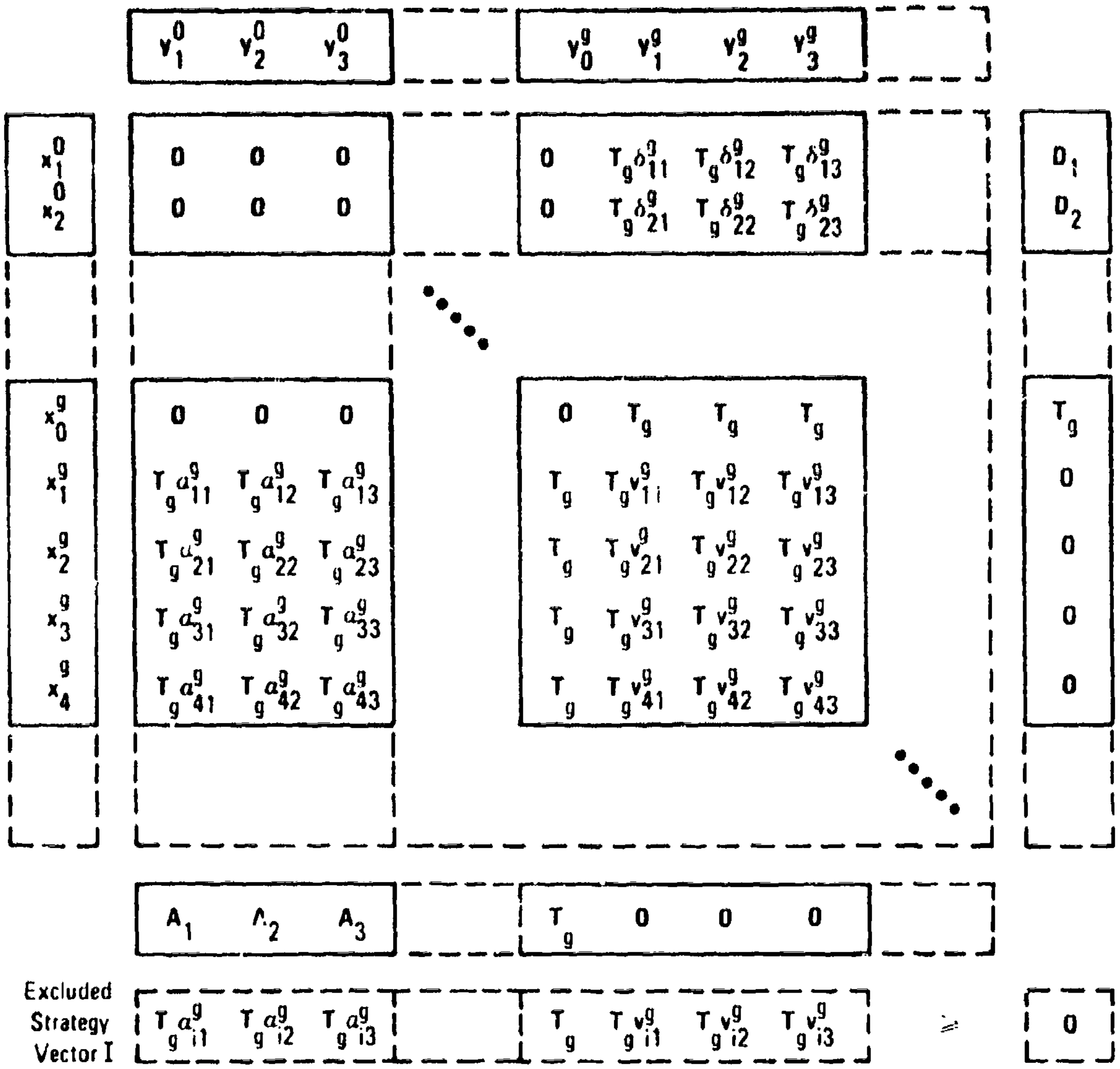
In Figure 3, X and Y are the augmented solution strategy vectors, Q is the matrix of coefficients of the basis equations, and A and D are the right-hand vectors. In matrix notation, the basis equations are $QY = D$ and $XQ = A$.

Figure 4 illustrates the partitioning of the Q-basis into blocks. Superscripts indicate the object type to which a block pertains, with a superscript zero indicating blocks associated with resources. There is a band of resource-associated blocks across the top of X, Q, and D and a band down the left side of Y, Q, and A. Down the main diagonal of Q is a line of object-type blocks. The rest of Q consists of zero elements.

Figure 5 illustrates the elements within typical resource blocks and object-type-g blocks. For convenience, the included strategies are given new i and j indexes to match their positions in Q, regardless of what i and j indexes they might have had originally. The elements shown as zero are always zero because of their positions; other elements will be zero only if one of their variable factors is zero.

At the bottom of Figure 5 is shown a scheme for relating the conditions on the excluded strategies to the structure of the Q-basis. If the vector shown is denoted by I, then the matrix condition is $IY \geq 0$. In similar fashion, an excluded maximizing strategy can be represented by a vector J and the exclusion condition by $XJ \leq 0$.

The various blocks of the Q-basis must be conformable as illustrated, but there is no a priori restriction on their size, with the exception that every Q^g block must be at least 2 x 2, since at least one elementary strategy for



y_1^0 y_2^0 y_3^0
y_0^g y_1^g y_2^g y_3^g
x_1^0 x_2^0
0 0 0
0 0 0
0 $T_g\delta_{11}^g$ $T_g\delta_{12}^g$ $T_g\delta_{13}^g$
0 $T_g\delta_{21}^g$ $T_g\delta_{22}^g$ $T_g\delta_{23}^g$
D_1 D_2
x_0^g x_1^g x_2^g x_3^g x_4^g
0 0 0
$T_g a_{11}^g$ $T_g a_{12}^g$ $T_g a_{13}^g$
$T_g a_{21}^g$ $T_g a_{22}^g$ $T_g a_{23}^g$
$T_g a_{31}^g$ $T_g a_{32}^g$ $T_g a_{33}^g$
$T_g a_{41}^g$ $T_g a_{42}^g$ $T_g a_{43}^g$
0 T_g T_g T_g
T_g $T_g v_{11}^g$ $T_g v_{12}^g$ $T_g v_{13}^g$
T_g $T_g v_{21}^g$ $T_g v_{22}^g$ $T_g v_{23}^g$
T_g $T_g v_{31}^g$ $T_g v_{32}^g$ $T_g v_{33}^g$
T_g $T_g v_{41}^g$ $T_g v_{42}^g$ $T_g v_{43}^g$
T_g 0 0 0 0
A_1 A_2 A_3
T_g 0 0 0
Excluded Strategy Vector I
$T_g a_{i1}^g$ $T_g a_{i2}^g$ $T_g a_{i3}^g$
T_g $T_g v_{i1}^g$ $T_g v_{i2}^g$ $T_g v_{i3}^g$
0

FIGURE 5
TYPICAL BLOCKS

each player is required, even if it is only the null strategy. Otherwise, Q^g may have any dimensions and may be square or rectangular, and if rectangular its long dimension may be in either direction. In general, different object types will have Q^g blocks of different size and shape.

Likewise, the total Q matrix itself may be either square or rectangular, but it is nearly always square, since if Q were to be rectangular, one player would have more variables in his augmented strategy than conditions for them to satisfy. In that case he could reasonably adjust to a better strategy restoring the balance between variables and conditions.

Detailed computations, involving the Q-basis elements, show that the value of the game is

$$\begin{aligned} V^* &= - AY^* \\ &= - X^*D \quad . \end{aligned}$$

Similarly, the minimax conditions can be verified by using the side conditions on excluded strategies. Furthermore, if Q is square and non-singular, then

$$X^* = AQ^{-1} \quad ,$$

$$Y^* = Q^{-1}D \quad , \text{ and}$$

$$V^* = - AQ^{-1}D \quad .$$

These formulas provide a method for computing the solution if the Q-basis is known. Section III will describe a method of finding the basis.

Before leaving the discussion of solution strategies, their physical feasibility should be discussed. The components, x_i^g and y_j^g, are fractions by definition. In general, the products, $T_g\ x_i^g$ and $T_g\ y_j^g$, will still be fractions and hence not physically feasible. Yet, if the solution strategies are interpreted as mixed strategies in a game-theoretic sense, it is usually possible to find not only one, but many, equivalent sets of physically feasible integral strategies, with an associated frequency for each strategy of the set.

In another view, the fractions, x_i^g for example, may be considered as a frequency distribution on each object of type g, with the important proviso that the distributions on the several objects of type g are not independent of each other or of the distributions on objects of other types.

III. PATH METHOD

The path method of solution is a form of parametric linear programming involving variation of resources. In this section we will describe how, as resources are varied, a Q-basis can be changed so that it will always provide a solution of the problem for the current numbers of resources.

A. Space of Resources

It is helpful to think of any set of resources

$$(A_1, \cdots, A_M; D_1, \cdots, D_N)$$

as defining a point in an (M + N)-dimensional space of resources. Associated with each point of this space is a distinct problem, or matrix game, which has at least one pair of solution strategies, X* and Y*, and a unique value of the game, V*.

The values, V*, may be thought of as plotted in (M + N + 1)-dimensions so as to form a continuous solution surface over the space of resources. At nearly every point of this surface there is a unique tangent hyperplane, whose slopes in the coordinate directions are the same as the marginal values defined in Section II, i.e.,

$$(-y_1^o, \cdots, -y_M^o; -x_1^o, \cdots, -x_N^o)$$

For each point of the resource space there is at least one valid Q-basis from which solution strategies and the value of the game may be computed. There may be more than one such valid Q-basis, each yielding different solution strategies, but there is only one value of the game at any point.

For nearly every point of the resource space, a valid Q matrix is square and non-singular. Such a basis is usually valid for a continuous set of points that we will call a region of the resource space. Throughout a region, the strategies are uniquely defined by the basis.

The interior of a region is generally characterized by proper inequalities:

$$x_i^g > 0 \text{ for included } i$$
$$y_j^g > 0 \text{ for included } j$$
$$IY > 0 \text{ for excluded } i$$
$$XJ < 0 \text{ for excluded } j \quad .$$

The boundary of a region is characterized by one or more of these quantities becoming equal to zero.

B. The Path

The continuous variation of resources from some initial set of values in some prescribed fashion to some terminal set of values generates a path through the resource space. The path method enables us to find solutions all along a fairly arbitrary path if we have a solution basis at the initial point.

There is one point in every type of problem where the solution basis is always known. That point is the origin, where all resources are zero and both players use null strategies. Hence, a path can always start at the origin, as it does in the computer programs of Sections IV and V. A path may, however, start at any other point where the Q-basis is known.

The simplest kind of path is generated by varying only one resource type at a time, all the others being held constant. We will call this a rectangular path, since it consists of a series of straight-line segments at right angles to each other in the resource space. This kind of path is used in the programs of Sections IV and V. In this method, the dimensions of the problem start at $(M,N) = (0,0)$ and increase as resource types are introduced. Once a type has been introduced, the number of resources of that type may be increased or decreased at will.

A more general kind of path is one consisting of straight-line segments generated by simultaneously varying any or all of one player's resources linearly in terms of some parameter. A segment of such a path can be represented by a set of equations of the form

$$A_m = A_{0m} + A_{1m} h, \quad 0 \le h \le \bar{h} \quad ,$$

expressing the condition that the number of type m resources varies from an initial value of A_{0m} when the parameter $h = 0$ to a terminal value of $A_{0m} + A_{1m} \bar{h}$. In similar fashion, a variation of maximizing resources can be expressed as

$$D_n = D_{n0} + D_{n1} h \quad .$$

The parameter h may be arbitrary or it may have some real meaning, e.g., a vehicle that carries fixed numbers of each resource type, or a budget allocated in fixed proportions to the different resource types. Either minimizing or maximizing resources may be varied on a path segment, but we have no convenient method of varying both simultaneously.

C. Location of a Regional Boundary

If a path segment lies entirely in a single region, the same Q-basis provides a solution at every point of the segment. If not, the first problem is to find the value of h for which the path meets a regional boundary. This critical value of h defines a critical point on the path where some changes must be made in the Q-basis before proceeding.

To do this for a minimizing path, for example, we first add a row to the A vector of Figure 4 to reflect the variable nature of the resources, i.e., A becomes the two-row matrix

$$A = \begin{vmatrix} A_{01}, & \cdots, & A_{0M}, T_1, 0, & \cdots, & T_g, 0, & \cdots, & 0 \\ A_{11}, & \cdots, & A_{1m}, 0\ , 0, & \cdots, & 0\ , 0, & \cdots, & 0 \end{vmatrix}$$

assumed to be multiplied by $H = (1, h)$. The matrix equation of condition then becomes

$$XQ = HA \quad .$$

The vector solution

$$X = HAQ^{-1}$$

is then equivalent to a set of scalar equations of the form

$$x = x_0 + x_1 h$$

One way the basis can fail is by some x_i^g for included i becoming negative. If $x_1 \geq 0$, x cannot become negative as h increases and we pass on to the next x. If $x_1 < 0$, we solve the equation

$$h = -x_0/x_1$$

to find the value of h for which $x = 0$. The least of these values of h is a candidate for the critical value.

The second way the basis can fail is for XJ to become greater than 0 for some excluded j. The scalar product XJ is of the form

$$x_0 + x_1 h \quad .$$

Applying a two-part test for every excluded j, we pass on to the next if $x_1 \leq 0$, since then x cannot become positive. If $x_1 > 0$, we solve the equation

$$h = -x_0/x_1$$

to find the value of h for which XJ = 0. The least of these values is a second candidate for the critical value.

The lesser of the two candidates is the critical value of h, which by substitution defines the critical point on the path and the solution strategy for the minimizing player at that point.

If the critical value comes from $x_i^g = 0$, then the corresponding row of the Q-basis must be deleted; if it comes from XJ = 0, then a corresponding column must be added to the Q-basis. In either case the resulting Q-basis will have one more column than it has rows.

In similar fashion, with appropriate formulations and test criteria, we can find the critical value along a path in the space of resources of the maximizing player; in that case, the resultant change in the Q-basis is deletion of a column or addition of a row, and the Q-basis will have one more row than it has columns.

D. Finding the Q-Basis for the Next Region

The rectangular Q-basis resulting from the location of a boundary is a valid basis on the boundary. Taking the example of a minimizing path, there is one more column than row, and hence one more condition on the minimizing player than he has variables. Although the extra condition is redundant and is satisfied by the minimizing player's solution strategy at the critical point, it does act as an added constraint.

On the other hand, the maximizing player here has one more variable than conditions to be satisfied. Because of this, there is an infinite number of solutions for the maximizing player. In fact, any linear combination of his solutions on the two sides of the boundary is a solution for him on the boundary.

It is the degree of freedom of the rectangular Q-basis that lets the maximizing player shift his strategy on the boundary to that strategy appropriate in the next region. We determine the new strategy by means of a new parameter, the marginal value per unit of the minimizing resource parameter and denote the new parameter by e, defined as

$$e = -A_{11}y_1^o - \cdots - A_{1M}y_M^o \quad .$$

We represent this condition by temporarily adding in the last row of the Q matrix the vector

$$(A_{11}, \cdots, A_{1M}, 0, \cdots, 0) \quad ,$$

adding to the D matrix a second column consisting of all 0s except for -1 in the last row, and assuming D to be multiplied by the vector $E = \begin{vmatrix} 1 \\ e \end{vmatrix}$. The matrix equation of condition becomes

$$QY = DE$$

The vector solution

$$Y = Q^{-1}\, DE$$

is equivalent to the set of scalar equations of the form

$$y = y_0 + y_1 e \quad .$$

The maximizing player is interested in driving e to a value as high as possible algebraically, that is in maximizing survival in the direction of the path. But he is limited by the conditions, $y_j^g \geq 0$ for every included j, and $IY \geq 0$ for every excluded i. The critical value of e is the lowest value meeting one of these conditions.

Applying a two-part test to y, we pass on if $y_1 \geq 0$, since y cannot then become negative as e increases. If $y_1 < 0$, we solve the equation

$$e = -y_0/y_1$$

to find the value of e for which x = 0. The least of these values of e is a candidate for the critical value.

The second test is on the condition $IY \geq 0$. This product is of the form

$$y_0 + y_1 e \quad .$$

We pass on to the next i if $y_1 \geq 0$, since y cannot then become negative as e increases. If $y_1 < 0$, we solve

$$e = -y_0/y_1$$

and select the least of these values as a second candidate for a critical value.

The lesser of the two candidates is the critical value of e, which by substitution defines the maximizing strategy along the path into the next region.

The rectangular Q-basis is then changed by deleting the column or adding the row associated with critical e. This change restores Q to a square, non-singular matrix, the solution basis for the next region.

In similar fashion, with appropriate formulations and tests, we find the Q-basis for the next region along a maximizing path.

E. Terminating a Path Segment

The two processes described in Sections III.C. and III.D. repeat in alternation until the end of a path segment is signaled by critical $h \geq \bar{h}$. At this interior point of a region, the Q-basis is not changed, the terminal strategy can be determined by substitution of $\bar{h}$, the terminal coordinates are entered in the first row of A or first column of D, as the case may be, and either a new path segment is begun or the problem is ended.

F. Beginning a Path Segment

The previous exposition was given with M + N resource types being allocated. If the new path segment specifies a variation of no more than these resource types, then the terminal point of the previous segment is an interior point of the space and we begin the new path by finding a boundary as in Section III.C. above.

If, however, a new type of resource is introduced, the terminal point of the previous segment automatically becomes a boundary point of the new (M + N + 1)-dimensional space of resources. The Q-basis must be expanded by opening a row or column of zeros at the appropriate place. The new path must then be begun by using the process described in Section III.D.

G. Beginning a Problem

Since the method will not permit introducing more than one type of resource at a time, it is usual to begin a problem at the origin, where both minimizing and maximizing players are constrained to a null strategy and where the Q-basis has 2G rows and columns as shown in Figure 6.

The problem is then begun by introducing one resource type in any desired numbers, followed by the introduction of other resource types until all M + N have been introduced.

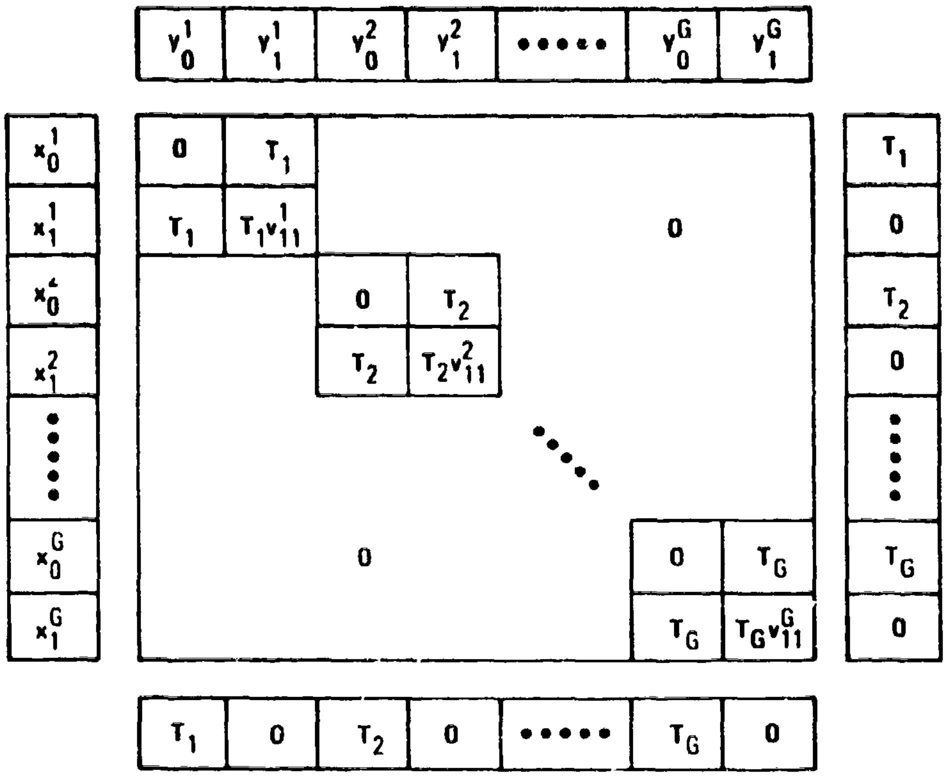


FIGURE 6
Q-BASIS AT THE ORIGIN

The order of introduction may make some difference in the computational time but makes no difference in the terminal value of the game. However, there are certain simple restrictions that must be observed. For example, the marginal value must not be equal to zero. This would be the case in the weapon allocation problem if we tried to introduce defense before attack, since in that case all targets would survive regardless of whatever allocation the defense might make.

IV. PATH87 COMPUTER PROGRAM

The PATH87 computer program is explained in this section. It is specifically designed to solve the multi-dimensional two-sided weapon allocation game for point targets. However, it can be modified with little trouble to solve a variety of resource allocation problems. In fact, ease of modifications has been one of the major criteria influencing program design.

The program is written in BASIC language for use on an IBM 360/65 computer in a time-sharing mode with interaction of computer and operator. The computer-system constraints that have been binding at one time or another during the evolutionary development of the program are:

-1 A limit of 800 statement lines

-2 A limit of 80 FOR loops

-3 A limit of 29 numeric arrays

-4 Limited storage space for the array elements of the problem.

The first three of those constraints have been overcome by such devices as using the same subroutines to serve both the minimizing and the maximizing player and using the same arrays to store similar numbers associated with both, thus taking advantage of the structural symmetry of the problem and effectively transposing large matrices back and forth by just changing a few indices.

The effect of the fourth constraint has been minimized by using a subroutine to compute the value function as needed instead of precomputing and storing it. Also, some matrices have been reduced in size by packing the significant elements.

Preceding page blank

Computer processing time has been reduced by designing a three-stage solution process with recursions. Further improvements in processing time have resulted from the use of single indices instead of double for nearly all arrays.

The price paid for the increase in capacity and decrease in running time has been some rather complicated indexing. However, even that is not completely without value, since the indexing is an aid to flexibility.

A. Program Structure

The program is composed of an initialization section, a control section, and a collection of subroutines. The heart of the program is the control section, which calls the main subroutines as needed.

Blocks of statement lines are allotted as follows:

(1-999)	Initialization section
(1000-1999)	Control section
(2000-2999)	Test subroutine
(3000-3999)	Auxiliary subroutines
(4000-4999)	Value subroutine
(5000-5999)	Print subroutine
(6000-6999)	Addition subroutine
(7000-8159)	Deletion subroutine
(8160-8999)	Auxiliary subroutines
(9000-9999)	Strategy subroutine.

The remainder of this section contains a complete listing of the program statements in numerical order, with explanatory comments following each small group of statements.

B. Initialization Section

This section sets the dimensions of the problem, initializes various indices, inputs data, and makes preliminary computations.

```
[illegible]

50 DATA 2,1,2,2
60 DATA 500,1,500,1
70 DATA .5,1,.5,1
80 DATA 1,.75
```

These lines provide an example of the data that an operator must type for any problem he proposes to run. Other examples appear in Section VI, Cases 5 to 8.

The first four numbers define the dimensions of the problem. In this example, there are 2 target groups (object types), of which 1 is defended, 2 attack-weapon types, and 2 defense-weapon types. The program later reads these numbers as G1, G4, A1, and D1, respectively. By convention, the defended target groups are the first G4 groups.

The next 2*G1 numbers define target data, that is the number of targets and value per target for each group.

The next G1*A1 numbers define attack-weapon data as a matrix of single-shot kill probabilities, in order of all weapon types against the first target group, then all weapon types against the second target group, etc. The weapon types must be listed in the same order for each target group, the preferred order being from least effective weapon to most effective weapon, as explained in Section IV.F.

The last D1 numbers define defense-weapon data as single-shot probabilities of intercept, it being assumed that these are the same against every type of attack weapon. The preferred order of listing is from most effective defense weapon to least effective, as explained in Section IV.F.

```
100 DIM G(100),T(100),V(100),P(2020),N(2020),Q(300)
105 DIM D(15),C(15),F(15),$(15),U(15,15),V(15,15),Z(15,15)
110 DIM I(40),K(15),L(15)
115 DIM X(250),Y(250)
120 DIM R(900),S(900),O(400)
125 DIM A(4000),F(1500)
```

These lines reserve storage space in core for 23 lists and arrays, those with related dimensions appearing on the same line. The reservations are adequate for most problems involving no more than 100 target groups (G1) and no more than 15 weapon types in all (A1 + D1).

All of the matrices will now be discussed, in a convenient order. The form (T) will denote a matrix itself, and the forms T(2), T(G), $(A1 + 1), etc., will denote particular elements of a matrix.

(T) stores the number of targets in each group. It is customarily addressed by the simple variable G, i.e., T(G).

(V) stores the value per target in each group and is usually addressed as V(G).

(P) stores the probabilities of kill. Conceptually, (P) is a two-dimensional array, but it is treated by the program as a one-dimensional list. The manner of address is discussed in Section IV.F. in connection with the value function, SUB 4000.

(D) stores the complements of the probabilities of intercept. The manner of address is discussed in connection with the value function.

($) stores the force levels attained at the end of any path segment. Elements $(1) to $(A1) represent levels of attack weapons, and $(A1 + 1) to $(A1 + D1) levels of defense weapons.

(A) is the largest matrix in the program. Primarily, it stores those elementary strategies of both attack and defense that are in the basis at any time. It also stores some other numbers related to those elementary strategies. Conceptually, it has an internal structure that is illustrated by the storage map of Figure 7. The matrix is partitioned into three main cell regions: test (1-60), attack (61-3000), and defense (3001-4000). The test region will be discussed in connection with the testing process, SUB 2000. The attack region is subdivided into rows, each row having A1 + 1 elements. The example showing 6 elements is based on A1 = 5. These rows are grouped. The first group, consisting of a single row, stores the identifying numbers of the attack weapons currently in play, e.g., 2, 3, and 5. The second group of G1 rows stores test control data derived from the elementary strategies. The remaining rows store the elementary strategies for each target group in vector form. For example, the second elementary strategy on target group 3 has 0 type-2 weapons, 2 type-3 weapons, and 1 type-5 weapon, for a total of 3 weapons. The maximum entries that have occurred in any elementary strategy row of target group 3 during the course of the run are stored in the third row of the test control group, e.g., 2, 2, 2, and 4. The defense region is structured and used like the attack region, but the rows have D1 + 1 elements, three in the example. The address of an element of (A) is usually compounded from its place in its row and the address of the terminal element of the preceding row, e.g., A(120 + 4) = 3.

Test	1	2	3	4	5	6	7	30
Best Test	31	32	33	34	35	36	37	60
Attack Weapon Number	61 2	62 3	63 5	64	65	66		
Test Controls	67 —	68 —	69 —	70 —	71	72		
	73 —	74 —	75 —	76 —	77	78		
	79 2	80 2	81 2	82 4	83	84		
G = 1 Strategies	85 —	86 —	87 —	88 —	89	90		
	91 —	92 —	93 —	94 —	95	96		
	97 —	98 —	99 —	100 —	101	102		
G = 2 Strategies	103 —	104 —	105 —	106 —	107	108		
	109 —	110 —	111 —	112 —	113	114		
G = 3 Strategies	115 0	116 0	117 2	118 2	119	120		
	121 0	122 2	123 1	124 3	125	126		
	127 1	128 1	129 1	130 3	131	132		
	133 2	134 1	135 1	136 4	137	138		
Defense Weapon Number	3001	3002	3003					

FIGURE 7
STRATEGY MATRIX (A)

(X) and (Y) store computed solutions of the parametric equations of the Q-basis. (X) stores the first column of a solution and (Y) the second column, i.e., the column which is multiplied by the parameter. They are used together, sometimes for an attack solution, sometimes for a defense solution. Whichever the case, the elements of (X) and (Y) are arranged in the appropriate sequence established in Section II and illustrated in Figure 5: marginal values for opposition weapons, intercept value on group 1, elementary strategies on group 1, ···, intercept value on group G1, elementary strategies on group G1.

Before discussing the other matrices of the program, it is desirable to describe in general terms a three-stage solution process that saves storage space and computer running time. Instead of storing and inverting a Q-matrix whose dimensions would exceed 200 x 200 if there were 100 target groups, we allow the Q-basis of Sections II and III to exist only as a mental concept of the equations of the problem. These equations are then solved by two stages of Gaussian elimination, a third stage in which a small matrix is inverted, and a back solution process which computes (X) and (Y) as full-size solutions of the conceptual Q-basis. Intermediate results are stored in such form that they can be modified recursively, without having to repeat the entire three-stage process every time a line is deleted from or added to the conceptual Q-basis. In the program, the matrices (Q), (R), (S), (U), (W), (Z), (C), and (F) are primarily involved in the process. These will now be discussed.

(R) stores the results of the first-stage Gaussian elimination, which reduces the Q-basis by 2 rows and 2 columns for each target group. The process is precisely defined. In the example of Figure 5, the upper left corner of the central block has this configuration--

$$\begin{matrix} 0 & \enclose{circle}{T_g} \\ \enclose{circle}{T_g} & T_g v^g_{11} \end{matrix} \quad .$$

Pivots on the circled elements eliminate two rows and columns. Because of the choice of pivot elements, the computations involve only subtractions, and very little information is lost by physically eliminating the pivot rows and columns. In fact, since the elementary strategies are preserved in (A) and the number of targets in (T), we need only provide for saving v_{11} elsewhere. (R) must be large enough to hold the elements not physically eliminated. Its size varies during a run. At the beginning, or at the origin, (R) is zero since all rows and columns are eliminated (see Figure 6). Then, as rows and columns are added to and deleted from the conceptual Q-basis, (R) is modified to reflect the changes. Suitable recursion algorithms are in the program for that purpose.

(Q) stores the values v_{11} for each target group. These "base" values would otherwise be lost after the first-stage elimination.

(S) stores the results of the second-stage Gaussian elimination, but preserves the physical configuration of (R). In concept, both (R) and (S) are rectangular, as illustrated by the pattern of Figure 8. However, the elements are stored according to the numbering scheme in the figure. These matrices are augmented to include elements from the right-hand sides of the equations of the problem. So the first column and first row of Figure 8 contain elements from vectors labeled D and A in Figure 3. The remaining

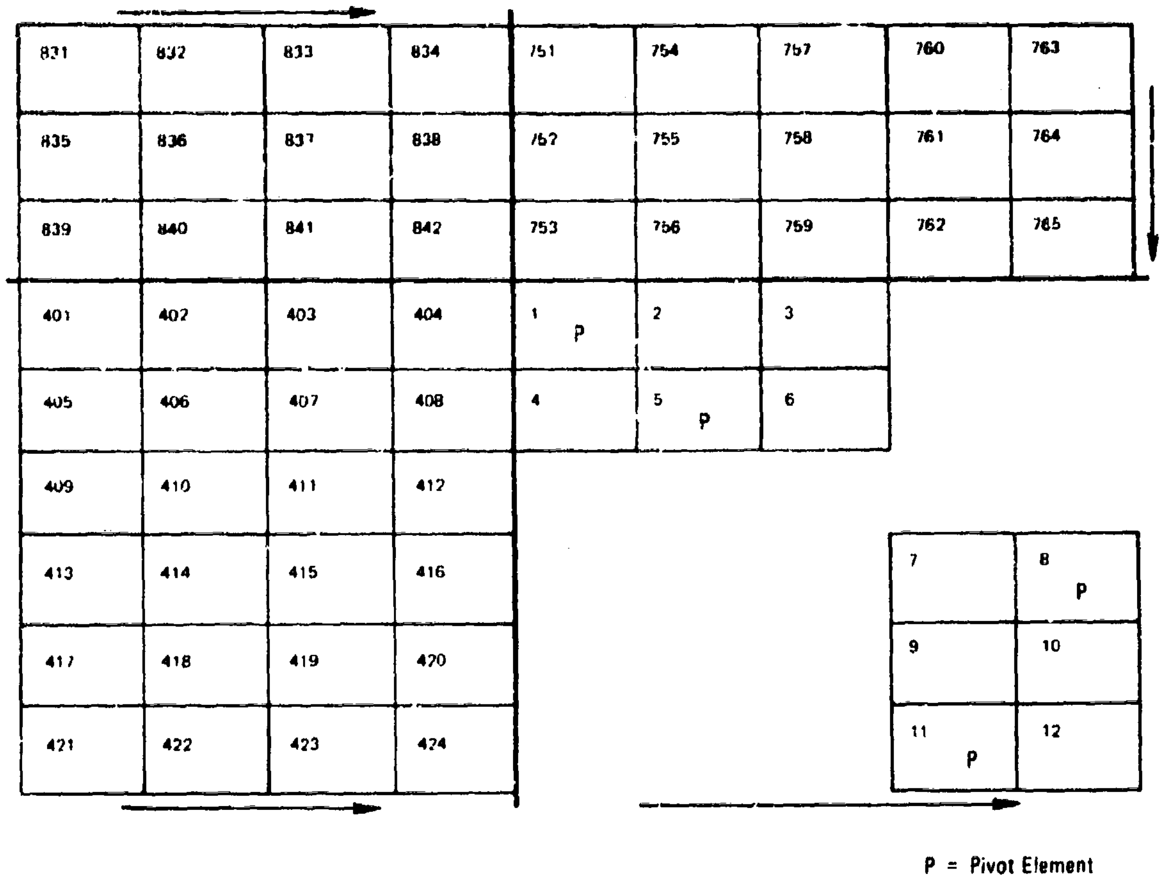


FIGURE 8
STORAGE MAP OF MATRICES (R) AND (S)

columns and rows correspond to the noneliminated columns and rows of the Q-basis. In this example, the columns represent 3 attack weapon types and 5 excess elementary defense strategies, and the rows represent 2 defense weapon types and 6 excess elementary attack strategies. The lower right region of Figure 8, which we will call the pivot region, contains elements left over after the first-stage elimination of two rows and columns from each of the blocks on the diagonal of the Q-basis. In this example, supposing there are 3 target groups in the problem, the gap in the pivot region implies that the original Q-basis block for group 2 has 3 rows but only 2 columns, and hence is not explicitly represented in (R) and (S). The blank elements in the pivot region are understood to be zeros, which need not be stored. The second-stage Gaussian elimination may be described as a partial inversion in place, since some of the procedures are adapted from the MATINV subroutine (Reference 6) for a total inversion in place. In concept, (R) is the given matrix and (S) is the result of the partial inversion. As many as possible pivot elements, such as those identified in Figure 8, are selected from the pivot region only, so that each pivot affects only one of the target groups. There is no shifting of rows or columns, but the location of pivots is recorded. The pivot rows and columns are actually used to store the results of the partial inversion, but it is understood that a pivot row or column has an alternative aspect, characteristic of a Gaussian elimination, in which the pivot element is 1 and all the other elements of the row or column are 0. The proper aspect is used at any point of the program. Finally, as lines are changed in the conceptual Q-basis, recursion algorithms make the induced changes in (S). At the end of the second stage, the number of equations remaining to be solved is somewhat unpredictable, but in theory can be bounded: Max {A1,D1} $\leq$ number $\leq$ A1 + D1. In the example of Figure 8, four equations remain to be solved out of an original fourteen in the conceptual Q-basis.

(U), (W), and (Z) are used in the third stage, which solves the remaining equations by matrix inversion. On each occasion, they are redimensioned by the program to precisely the size needed, e.g., in the continuation of Figure 8 they are redimensioned as 4 x 4 matrices. (U) is then filled with the elements of (S) that are in neither a pivot row nor a pivot column, as in Figure 9.

836	837	838	758
840	841	842	759
410	411	412	0
418	419	420	0

FIGURE 9
FORMATION OF MATRIX (U) FROM ELEMENTS OF MATRIX (S)

(W) stores the inverse of (U). (Z) stores either a duplicate or the transpose of (W), whichever is needed at the time. These three matrices are the only ones with two subscripts.

(C) and (F) store the two-column solution of the third-stage equations. Their elements become elements of (X) and

(Y), respectively, and are used in the back solution to complete these vectors.

(G) is the first of seven index matrices that are needed to help locate numbers in the primary matrices already discussed. (G) stores the locations in (S) of the initial elements of blocks of numbers pertaining to the different target groups. (G) consists of three regions, each having G1 + 1 elements. The first region contains the locations of blocks in the pivot region of (S). For the example of Figure 8, G(1) = 1, G(2) = 7 since this is where the block for the second target group would appear if it existed, G(3) = 7, and G(4) = 13, in effect defining an upper bound on the pivot region. Similarly, the next four elements locate blocks in the lower left-hand region of (S), i.e., G(5) = 401, G(6) = 409, G(7) = 413, G(8) = 425. The last four elements locate blocks in the upper right region of (S), i.e., G(9) = 751, G(10) = 760, G(11) = 760, G(12) = 766. In a problem with only three target groups, the remaining elements of (G) would never be used.

(N) stores the number of rows and columns of each block in the pivot region of (S). (N) consists of two regions of G1 + 1 elements each, the first region for rows, the second for columns. So, when (S) is as shown in Figure 8, this index will be:

$$(N) = (2,1,3,0;3,0,2,0;0,\cdots) \quad .$$

Actually, the G1 + first element of each region is superfluous, but is retained for convenience in indexing (N) itself.

(M) stores the total number of rows and columns of the pivot region preceding each block of the pivot region of (S). (M) is arranged the same way as (N). For our standard example, we have:

$$(M) = (0,2,3,6;0,3,3,5;0,\cdots) \quad .$$

In this case, the first element of each region might be considered superfluous.

(Ø) stores the locations of the actual pivot elements in (S), identifying them by rows and columns rather than by a linear storage number. (Ø) is divided into two regions of equal size, in this case having 30 elements each. The first region of Ø contains an entry for each row of the pivot region of (S). If the row has no pivot element in it, the entry is 0. If the row has a pivot element in some column, the entry is the number of that column within the block of columns for the particular target group involved. For the example of Figure 8, we have Ø(1) to Ø(6) given as (1,2,0,2,0,1). The second region contains similar entries that identify a row for any pivot in a column, so we have Ø(31) to Ø(35) given as (1,2,0,3,1).

(I) is the central-index matrix. It stores a variety of constants and variables that will be individually discussed in connection with lines 140-215. Its conceptual structure, shown in Figure 10, reflects the paired nature of most of its elements. It has a central spine of elements identified (except for 1) by numbers of the form 3n, and two wings, of the forms 3n-1 and 3n+1. The elements of one wing are associated with rows of the problem and those of the other wing with columns. An interchange of wings has the effect of transposing the Q-basis of the problem.

(K) and (L) provide the mechanism for the transposition, one wing of (I) being read into (K) and the other into (L) by SUB 8690. A flag S1, whose value is set at +1 or -1, determines which wing is read into which matrix. Most of

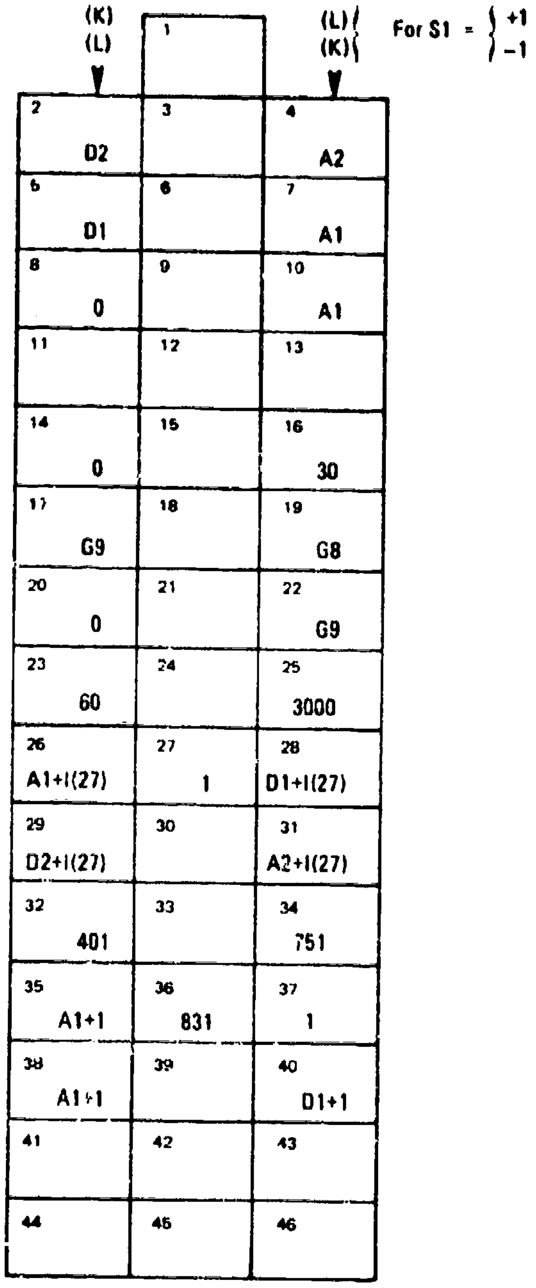


FIGURE 10
CENTRAL INDEX MATRIX (I)

the computational algorithms of the program are written in terms of (K) and (L) and are equally valid for row operations or column operations, that is for attack strategies or defense strategies.

```
130 READ C1,C4,A1,B1
135 PRINT "C1,C4,A1,B1=";C1;C4;A1;B1
```

These lines rcad and print the dimensions of the problem, i.e., the data of line 50.

```
140 C9,I(17),I(22)=C1+1
145 I(7),I(10)=A1
150 I(5)=B1
155 I(37)=1
160 I(16)=30
165 I(35),I(38)=A1+1
170 I(40)=B1+1
175 C8,I(19)=2*C9
180 I(23)=60
185 I(25)=3000
190 I(21),I(29),I(31),C(1)=1
195 I(26)=A1+I(21)
200 I(28)=B1+I(27)
205 I(32),C(C9+1)=401
210 I(34),C(C8+1)=751
215 I(36)=831
```

This sequence computes and initializes various indices, all others being automatically initialized at zero by the BASIC system. The paired structure of (I) is illustrated in the following discussion.

<u>K(1), L(1)</u>: I(2) is the number of types of defense weapons already brought into play, sometimes denoted by D2. I(4) is the number of types of attack weapons already brought into play, sometimes denoted by A2. Both I(2) and I(4) are variables, automatically initialized at 0 and stepped as new weapon types are added. K(1) and L(1) are used principally as upper limits for loops and as locators in (X) and (Y).

<u>K(2), L(2)</u>: I(5) and I(7) are constants, D1 and A1, respectively. They are not used in this version of the program.

<u>K(3), L(3)</u>: I(8) and I(10) are constants, 0 and A1, respectively. They are used as locators for the two regions of the ($) matrix.

<u>K(4), L(4)</u>: Not used in this version of the program.

<u>K(5), L(5)</u>: I(14) and I(16) are constants, 0 and 30, respectively. They are used as locators for the two regions of the (Ø) matrix or pivot index. In case (Ø) is redimensioned, for example to 80, then I(16) should be set at 1/2 Dim(Ø) = 40.

<u>K(6), L(6)</u>: I(17) and I(19) are constants, set at computed values G9 = G1 + 1 and G8 = 2*G9, respectively. They are used as locators for the two upper regions of the (G) matrix.

<u>K(7), L(7)</u>: I(20) and I(22) are constants, 0 and G9, respectively, serving as locators for the two regions of the (M) and (N) matrices.

<u>K(8), L(8)</u>: I(23) and I(25) are constants, used as locators for the attack and defense regions, respectively, of (A). In this version of the program they are set arbitrarily at 60 and 3000. The apportionment of space between attack and defense can be modified by changing I(25).

<u>K(9), L(9)</u>: I(26) and I(28) are constants, used to specify the number of elements of (A) needed to record the useful features of an elementary strategy for the attack or defense, respectively. In this version of the program the number of elements is one plus the number of weapon types, so that the number of weapons of each type plus the sum of the weapons of all types can be stored. The effect is to

structure the attack region of (A) into rows of size I(26) and the defense region into rows of size I(28).

K(10), L(10): I(29) and I(31) are variables, used to specify the number of elements currently in use in each of the rows just described. Both are initialized at the same value as I(27). Thereafter, I(29) is kept equal to D2 + I(27) and I(31) to A2 + I(27).

K(11), L(11): I(32) and I(34) are constants, locating the initial values of the α and δ regions of (S) respectively and are set at 401 and 751 in this version of the program. They may be changed if a different apportionment of storage space is desired.

K(12), L(12): I(35) and I(37) are constants, A1 + 1 and 1, respectively, used as stepping indices for the common region of (S). I(35) is the interval between elements in successive rows of the same column, and I(37) between elements in successive columns of the same row. Note that I(36), on the spine of (I), is the locator for the common region, here set arbitrarily at 831.

K(13), L(13): I(38) and I(40) are constants, A1 + 1 and D1 + 1, respectively. I(38) is the interval between elements in successive rows of the same column of the α region of (S) and I(40) is the interval between elements of successive columns of the same row of the δ region. These two are interchanged when (S) is transposed. They are used as stepping indices.

K(14), L(14), K(15), L(15): I(41), I(43), I(44), and I(46) are not used regularly in this version of the program. However, the spaces I(41) and I(43) are used to store strategy locators needed in SUB 4000. The spine of (I) is empty except for I(27) and I(36), which have already been discussed, and for I(1) and I(3) which are used as working storage for step indices in the pivot region of S. These vary from target group to target group. In SUB 8760, one of the pair is set at 1 and the other at N(G9 + G), these being the column-to-column and row-to-row steps for target group G, as shown in Figure 8.

```
300 PRINT "T,V="
310 FOR G=1 TO G1
320 READ T(G),V(G)
330 PRINT T(G);V(G)
340 Q(G)=V(G)
350 V1=V1+V(G)
360 G(G+1)=G(1)
370 G(G9+G+1)=I(32)
380 G(G8+G+1)=I(34)
390 NEXT G
```

This loop reads data on the number of targets T(G) and the value of each target V(G). It initializes Q(G) at the value V(G). The loop computes V1 = ΣV(G) as the simplest way of making V1 > max {V(G)} for use in the testing processes. Finally, it initializes the (G) locators to the (S) matrix, whose current dimensions are zero because there are no excess elementary strategies and no weapons to start with.

```
400 PRINT "PK1="
410 I=0
420 FOR G=1 TO G1
430 FOR A=1 TO A1
440 I=I+1
450 READ P(I)
460 PRINT P(I);
470 NEXT A
480 PRINT
490 NEXT G
```

This double loop reads the matrix of kill probabilities into (P) and prints it for reference.

```
500 PRINT "P.="
510 FOR D=1 TO D1
520 READ P
530 PRINT P;
540 D(D)=1-P
550 NEXT D
560 PRINT
```

This loop reads the intercept probabilities, prints them for reference, and stores the complements in (D).

C. Control Section

This control routine directs the computations over a path segment and is repeated for each segment of the path. It consists of four parts:

Segment initialization (1000-1380)

Strategy on boundary (1500-1590)

Strategy in region (1700-1810)

Segment termination (1820-1880).

The two middle parts are repeated in alternation as many times as required along the segment.

```
1000 PRINT "P1,P3?"
1010 INPUT P1,P3
1020 IF P1<>0 THEN 1040
1030 STOP
```

The path segment is defined by two numbers in this version of the program: P1, the identifying number of the single weapon type to be varied as parameter along the segment, and P3, the terminal number of weapons of that type. Here, the numbers are input by the operator, but the program can be modified to read the numbers as data or get them from a data file. Meaningful values of P1 are integers from -D1 to +A1, with defense weapon types identified by a - sign and the end of the program run signaled by a zero. P3 may have any non-negative value. Thus, the number of type P1 weapons may be increased or decreased along a path segment. However, a decrease to 0 should be avoided because of the possibility that round-off errors will throw the path into negative regions of the resource space. Special protections against this accident were written into an earlier version of the program, but have been eliminated in this version. If P1 = 0, the run stops; otherwise it continues.

```
1040 S1=SGN(P1)
1050 P1=ABS(P1)
1060 GOSUB 8690
1070 P2=SGN(P3-S(K(3)+P1))
1075 V2=V1*(1-P2)
1080 P4=ABS(P3-S(K(3)+P1))
1090 IF P4<>0 THEN 1130
1100 GOSUB 9290
1110 GOTO 1860
```

This sequence sets S1 at +1 or -1, converts P1 to a positive number, and goes to SUB 8690 to set the (K) and (L) indices associated with S1. It also sets P2 at +1, 0, or -1 depending on whether the number of weapons is to be increased, unchanged, or decreased from the number at the end of the preceding path segment; sets V2 at 0, V1, or 2V1 for later use in testing; and sets P4 at the amount of change in the number of weapons. If P4 = 0, the program goes to SUB 9290 to compute a strategy, and then branches to the segment termination, an option used when the operator has just completed a path segment, obtaining a printout of either the attack or defense solution strategy, and wishes to obtain a printout of the other solution strategy at the same point of the path. Otherwise, the run continues.

```
1130 FOR R5=1 TO L(1)
1140 IF A(K(8)+R5)=P1 THEN 1710
1150 IF A(K(8)+R5)>P1 THEN 1180
1160 NEXT R5
1170 R5=L(1)+1
```

This sequence examines the proper weapon-number row of (A) to find the proper column for weapon P1, designating this column by R5. If weapon P1 already has a column specified, initialization is completed, and the program branches to 1710. If not, then the column is picked that will put P1 in the proper sequence with the weapons already listed in (A). For example, if attack weapon 1 is to be added to Figure 6, R5 will be 1, and the program will continue at 1180.

```
1180 FOR R=L(1) TO R5 STEP-1
1200 I=I(36)+R*L(12)
1210 FOR K=0 TO K(1)
1230 S(I+L(12))=S(I)
1240 S(I)=0
1245 I=I+K(12)
1250 NEXT K
1254 I=K(11)+R
1256 FOR K=1 TO M(K(7)+69)
1260 S(I+1)=S(I)
1262 S(I)=0
1263 I=I+K(13)
1264 NEXT K
1270 NEXT R
```

This sequence opens up and zeros the R5 column of (S), shifting other columns as required. Statements 1200-1250 open a column (or row) in the common region, and statements 1254-1264 open a column (or row) in the α (or δ) region, depending on whether S1 = +1 or -1.

```
1280 FOR J=K(8) TO K(8)+K(9)*(M(K(7)+69)+61+61) STEP K(9)
1290 FOR L=J+L(10) TO J+R5 STEP -1
1300 A(L+1)=A(L)
1310 NEXT L
1320 A(J+R5)=0
1330 NEXT J
1340 A(K(8)+R5)=P1
```

This sequence opens up and zeros the R5 column of (A), and enters the number P1 in the weapon row of this column.

```
1350 R1=R1+1
1360 L(1),I(3+S1)=L(1)+1
1380 L(10),I(30+S1)=L(10)+1
```

This sequence steps two indices in (I) to reflect the addition of a weapon type, and also steps the variable R1, which measures the greater of the two quantities:

I(2) + Excess Attack Strategies

I(4) + Excess Defense Strategies.

R1 is automatically initialized at zero and then is increased or decreased as required. R1 is a measure of the size of (S), and is used in setting the current size of the third-stage matrix (U).

```
1500 S1=-S1
1510 S2=-1
1520 GOSUB 9000
1530 H2,H3=V2
1540 GOSUB 2000
1550 IF H2<=H3 THEN 1580
1560 GOSUB 6000
1570 GOTO 1700
1580 GOSUB 7000
1590 R1=R1-1
```

This sequence controls the computation of a strategy crossing a boundary (see Section III.D.), where the Q-basis is rectangular and the parameter is marginal value, as indicated by the flag S2 = -1. The flag S1 = 1 indicates an attack strategy, S1 = -1 a defense strategy. SUB 9000 computes (X) and (Y). SUB 2000 finds two candidates, H2 and H3, for a critical value of the parameter. These are compared and the program branches to SUB 6000 to add a row (or column) to the basis, or to SUB 7000 to delete a column (or row) from the basis. In the latter case, R1 is reduced because (S) has changed size. In either case, the new basis is square and the program continues at line 1700.

```
1700 S1=-S1
1710 S2=1
1720 GOSUB 9000
1730 H2,H3=P4
1740 GOSUB 2000
1750 IF H2<=H3 THEN 1790
1760 R1=R1+1
1770 GOSUB 6000
1780 GOTO 1500
1790 IF H2=P4 THEN 1820
1800 GOSUB 7000
1810 GOTO 1500
```

This sequence controls the computation of a strategy in the interior of a region or at a regional boundary (see Section III.C.), where the Q-basis is square and the parameter is resource variation, as indicated by the flag S2 = 1. The control sequence is the same as in the 1500 routine. However, H2 and H3 are initialized at P4 instead of at V2, R1 is increased if a row (or column) is added to the basis, and a test at line 1790 provides an exit to the sequence for terminating the path segment. Otherwise, the program continues at line 1500, which alternates with 1700 until the path segment is terminated.

```
1820 $(K(3)+F1)=F3
1840 I=I(36)+L(12)*F5
1850 S(I)=S(I)+F2*F4
1860 H=F4
1870 GOSUB 5000
1880 GOTO 1000
```

The path segment is terminated by posting the current force level in ($), adjusting a right-side element of (S), and printing the terminal strategy in SUB 5000. The program then goes back to line 1000 for the next path input.

D. Test Subroutine

The test subroutine finds two candidates for the critical value of a parameter, as discussed in Section III. Candidate values, H2 and H3, are initialized in the control program and modified during testing. H2 is associated with the non-negativity condition and H3 with the scalar-product condition. The final choice between candidates is made after return to the control routine.

The subroutine consists of an index section, a branch controlling tests on a defense strategy, and a branch controlling tests on an attack strategy. Seven auxiliary subroutines serve both control branches.

```
2000 I1=K(1)+1
2010 J0=I(23)+I(20)
2020 J1=K(8)+K(9)+G1
2030 I4,I(42+S1)=0
2035 J7=42-S1
2040 IF S1=1 THEN 2500
```

This sequence initializes indices for the first target group: I1 locates the intercept elements in (X) and (Y); J0 locates the control row of (A); J1 locates the base elementary strategy in (A); and I4 locates the row of (P) containing single shot probabilities of kill for targets of this group. In addition, I(42 + S1) is set at 0, and J7 is identified with 42 - S1 so that I(J7) may be used as a working strategy locator in connection with the value function, SUB 4000. After initialization, the program branches to one or the other of the teot-control sequences.

```
2100 FOR G=1 TO G1
2110 GOSUB 3000
2120 GOSUB 3200
2130 [illegible] 2190
2140 GOSUB 3500
2150 GOTO 2170
2160 K=1
2170 IF A(K)>A(J6+K) THEN 2230
2180 GOSUB 3600
2190 GOSUB 3700
2200 IF Y2>-.001 THEN 2220
2210 GOSUB 3300
2220 IF A(K(I0))<=A(J6+I(J4))) THEN 2160
2230 IF K<K(1) THEN 2140
2240 GOSUB 3820
2250 NEXT G
2260 RETURN
```

This sequence controls the testing when (X) and (Y) represent a defense strategy. The test sequence for each target group is illustrated by the flow diagram of Figure 11. SUB 3000 conducts the non-negativity tests for components of the defense strategy. The rest of the diagram is concerned with the scalar-product tests for elementary attack strategies not in the basis.

The general scheme is to generate an elementary attack strategy, compute the value function for its combination with each elementary defense strategy in the basis, compute the scalar product with (X) and (Y), and test for criticality. The generating scheme in this version of the program is what we call a "floating lid." It generates all strategies over a truncated rectangular region whose size is controlled by a test control row of the (A) matrix. In the example of

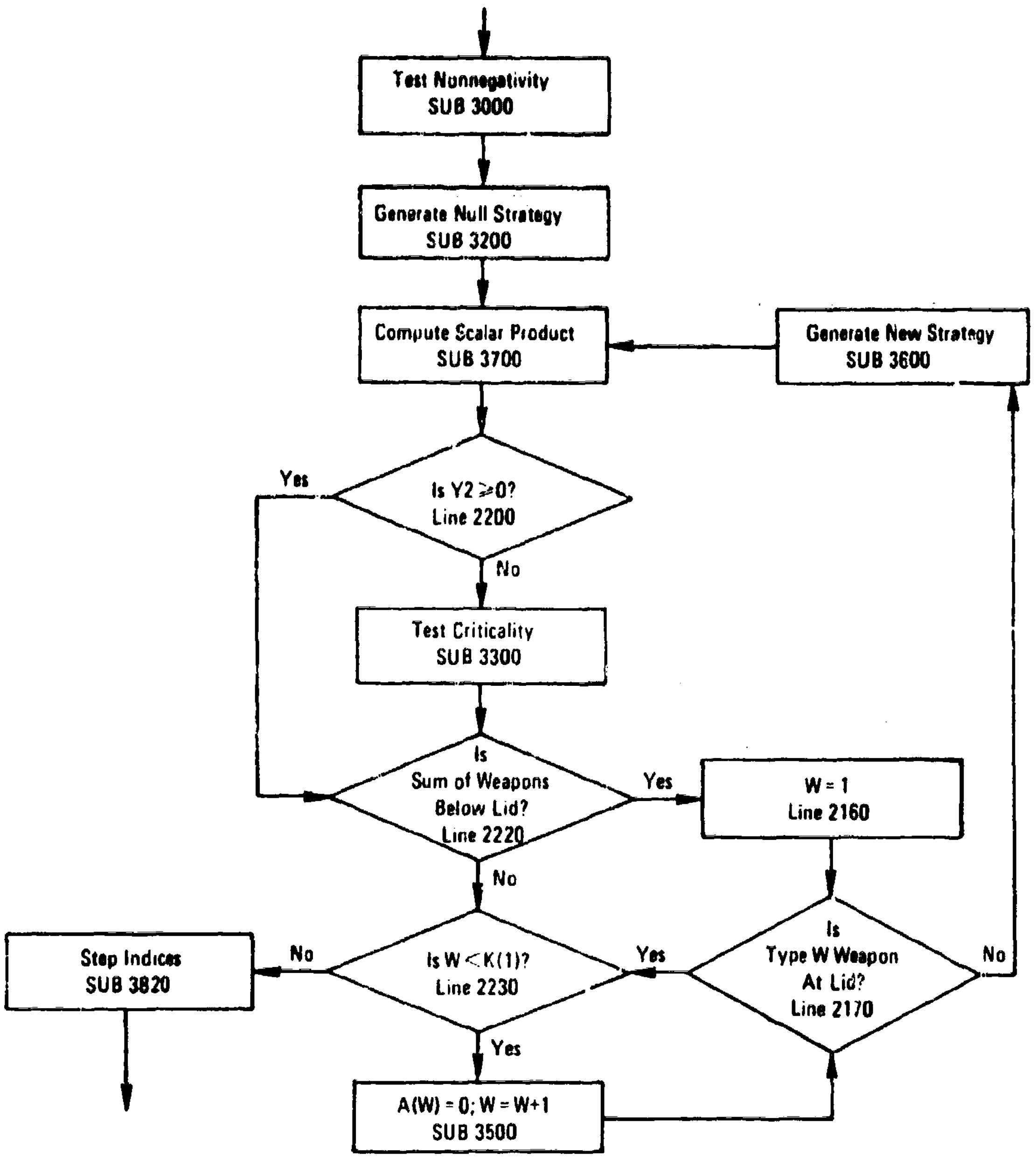

Test Nonnegativity SUB 3000
Generate Null Strategy SUB 3200
Compute Scalar Product SUB 3700
Generate New Strategy SUB 3600
Is Y2 ≥ 0? Line 2200
Yes
No
Test Criticality SUB 3300
Is Sum of Weapons Below Lid? Line 2220
Yes
No
W = 1 Line 2160
Is Type W Weapon At Lid? Line 2170
No
Yes
Is W < K(1)? Line 2230
No
Yes
Step Indices SUB 3820
A(W) = 0; W = W+1 SUB 3500

FIGURE 11
TEST SEQUENCE (Statement Lines 2110-2240)

Figure 7, the test control row for G = 3 consists of the numbers 2, 2, 2, 4. All test strategies are generated that would not increase any of those numbers by more than 1, specifically all strategies satisfying:

$$0 \le \alpha_2 \le 3 \quad ,$$
$$0 \le \alpha_3 \le 3 \quad ,$$
$$0 \le \alpha_5 \le 3 \quad ,$$
$$0 \le \sum_m \alpha_m \le 5 \quad .$$

The first three conditions define a rectangular region of 64 strategies, and the fourth truncates it so that only 44 test strategies are actually generated. The "floating lid" scheme is a compromise that gives excellent results for the two-sided game but may be slightly off optimum for one-sided attack allocations, especially if some of the weapon types have very low kill probabilities.

Test attack strategies are generated recursively in the 1-30 region of matrix (A), and the current critical candidate is stored in the 31-60 region of (A). The order of storage is α_m values, $\sum_m \alpha_m$, and computed v's, one v for each elementary defense strategy on the target group under test. The method of generation is by a nest of implicit loops on the weapon types currently in play.

The scalar product of the test-strategy vector with the solution vectors is represented as X2 + H*Y2, where the coefficients, X2 and Y2, correspond with (X) and (Y). For efficiency, partial sums are computed by recursion as X1 and Y1.

We can now describe the flow diagram of Figure 11 in more detail. SUB 3200 generates an initial test strategy with all zero elements. SUB 3700 computes values. If Y2 ≥ 0, the strategy cannot be critical; otherwise, SUB 3300 compares it for criticality. The A(K(10)) test determines if the "lid" on the sum has been reached. If not, the first weapon type is set by W = 1. If the lid has been reached and W < K(1), then SUB 3500 reduces A(W) to zero and sets W = W + 1. The A(W) test determines if the "lid" on weapon W has been reached. If not, SUB 3600 steps A(W) and the cycle repeats. If so, and if W = K(1), then testing on the group is finished, and SUB 3820 steps indexes for the next group.

```
2500 FOR G=1 TO G1
2510 GOSUB 3000
2520 IF I(2)=0 THEN 2660
2530 IF G>G4 THEN 2660
2540 IF A(JC+I(31))=0 THEN 2660
2550 GOSUB 3200
2560 GOTO 2610
2570 GOSUB 3500
2580 GOTO 2600
2590 W=1
2600 GOSUB 3600
2610 GOSUB 3700
2620 IF Y2<.001 THEN 2640
2630 GOSUB 3300
2640 IF A(K(10))<A(JC+I(31)) THEN 2590
2650 IF W<K(1) THEN 2570
2660 GOSUB 3820
2670 NEXT G
2680 RETURN
```

This sequence controls the testing process when an attack strategy is being determined. It is evident by comparison with (2100-2260) that the general scheme and computational subroutines are the same, but there are significant differences in the control processes. If no defense weapon has been introduced, if G is an undefended group, or if G has not been brought under attack, no tests of new defense strategies are appropriate. So, an immediate branch to 2660 occurs.

The range of elementary defense strategies to be tested is determined solely by the attack-summation test control number A(J0 + I(31)), and the sole cut-off control is statement 2640 which prevents defense use of more weapons than A(J0 + I(31)). In the example of Figure 7, defense strategies to be tested must satisfy the single condition:

$$0 \leq \sum_n \delta_n \leq 4 \quad .$$

If there were two types of defense weapons, this inequality would call for the generation of 15 elementary strategies.

E. <u>Auxiliary Subroutines</u>

```
3000 K4=N(K(7)+G)
3010 IF K4=0 THEN 3100
3020 FOR I=I1+1 TO I1+K4+1
3030 IF Y(I)>-.0001 THEN 3090
3040 H=-X(I)/Y(I)
3050 IF H2<H THEN 3090
3060 H2=H
3070 G2=G
3080 M2=I-1-I1
3090 NEXT I
3100 RETURN
```

This subroutine tests the condition $x_j^g \geq 0$ for elementary strategies in the basis. It solves for H, the value of the parameter that makes $x_j^g = 0$, and records H2, the candidate for critical value. It also records G2, the group in which that candidate is found, and M2, a locator for the particular elementary strategy among those of group G2.

```
3200 FOR K=1 TO K(10)
3210 A(K)=0
3220 NEXT K
3240 X1,X2=X(11)
3250 Y1,Y2=Y(11)
3260 RETURN
```

This subroutine generates the null strategy as an initial test strategy and initializes the scalar products.

```
3300 H=-X2/Y2
3310 IF H3<H THEN 3400
3320 H3=H
3330 G3=G
3370 FOR K=1 TO K(10)+K4+1
3380 A(K+30)=A(K)
3390 NEXT K
3400 RETURN
```

This subroutine computes the value of the parameter that makes the scalar product zero, compares this value with H3 and replaces H3 if H3 ≥ H; in this case, the subroutine records G3 = G and transfers all pertinent data from the 1-30 region of (A) to the 31-60 region, where the best test strategy is recorded. This completes the testing of one elementary strategy.

```
3500 X1=X1-A(W)*X(W)
3510 Y1=Y1-A(W)*Y(W)
3520 A(K(10))=A(K(10))-A(W)
3530 A(W)=0
3540 W=W+1
3550 RETURN
```

This subroutine reduces A(W) to zero after making related changes and steps W to W + 1.

```
3600 A(W)=A(W)+1
3610 A(K(10))=A(K(10))+1
3620 X1,X2=X1+X(W)
3630 Y1,Y2=Y1+Y(W)
3640 RETURN
```

This subroutine generates a new test strategy by stepping A(W) and making related changes.

```
3710 N=11
3720 I(J7)=J1
3730 FOR W=1 TO K4+1
3740 N=N+1
3750 I(J7)=I(J7)+K(4)
3760 GOSUB 4000
3770 A(K(10)+W)=V
3780 X2=X2+V*X(W)
3790 Y2=Y2+V*Y(W)
3800 NEXT W
3810 RETURN
```

This subroutine controls access to SUB 4000, the value function; it also sums for the coefficients, X2 and Y2, of the scalar product.

```
3820 I1=I1+K4+2
3825 J6=J6+I(26)
3830 J1=J1+K(4)*(K4+1)
3850 I4=I4+61
3860 RETURN
```

This subroutine steps indices for the next target group.

F. Value Subroutine

This subroutine is addressed from statement 3760 only. It computes and returns one number, V, the expected value surviving of a single target of type G when it is attacked using the elementary strategy located at I(41) in (A) and defended using the elementary strategy located at I(43) in (A), these two locations having been defined at statements 2030 and 3750, respectively.

In this version of the program, it is assumed that the firing sequence of attack weapons is worst to best with the idea of using poor weapons to exhaust interceptors, and the sequence of defense weapons is best to worst with the idea that any leftover interceptors will be the worst. This is the order in which weapons have been numbered in the original data input for the program. Interceptors and attacking weapons are thus matched in pairs, as is illustrated in Figure 12 for the strategies:

$$\vec{\alpha} = (3,2,3)$$

$$\vec{\delta} = (2,4) \quad .$$

In the illustration, the expected value surviving is:

$$\begin{aligned} V = V_g &\cdot [1 - PKT_1\ (1-PI_1)]^2 \\ &\cdot [1 - PKT_1\ (1-PI_2)] \\ &\cdot [1 - PKT_2\ (1-PI_2)]^2 \\ &\cdot [1 - PKT_3\ (1-PI_2)] \\ &\cdot [1 - PKT_3]^2 \end{aligned}$$

where PKT_i is the probability of target kill by a single attack weapon of type i and PI_j is the probability of intercept by a single defense weapon of type j. Subroutine 4000 computes V according to this scheme.

```
4000 V=V(0)
4010 D,I=0
4020 FOR A=1 TO I(4)
4030 A7=A(I(41)+A)
4070 IF A7=0 THEN 4270
4080 I5=I4+A(I(23)+A)
```

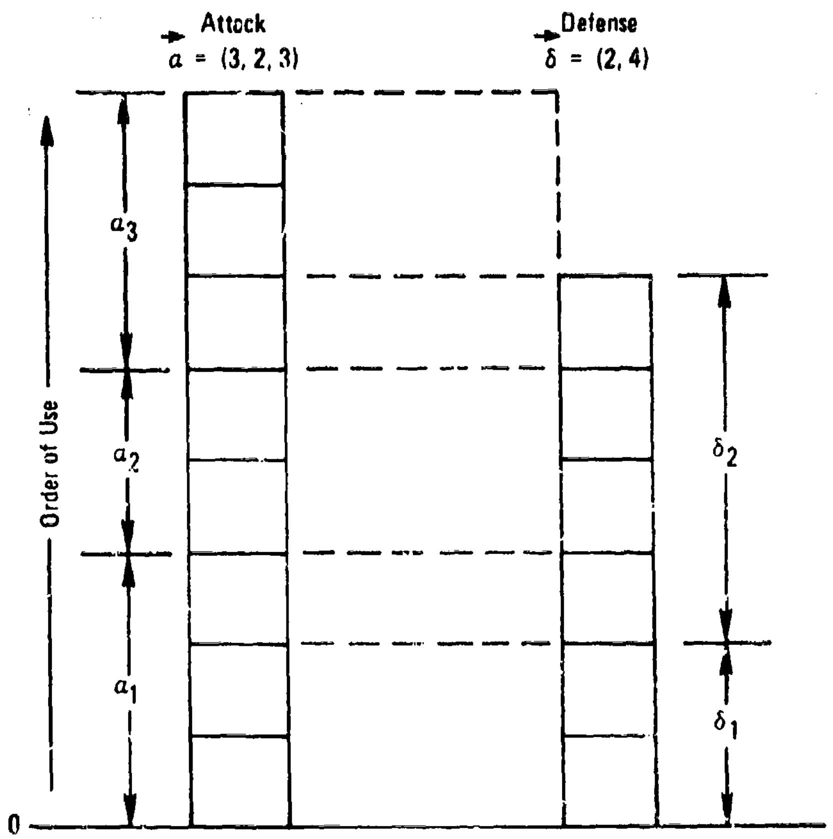


FIGURE 12
A PAIR OF ELEMENTARY STRATEGIES

This sequence initializes value, defense weapon type, and number of interceptors; begins a loop on A extending to line 4270; sets A7 = the number of type A weapons in the $\vec{\alpha}$ strategy; if A7 = 0, goes to the next A; otherwise, sets I5 to locate PKT for weapon type A in matrix (P).

```
4090 IF G>G4 THEN 4260
4100 IF I>0 THEN 4180
4110 IF D=I(2) THEN 4260
4120 D=D+1
4130 I=A(I(43)+D)
4170 GOTO 4100
```

This sequence decides whether an interceptor reading is in order; if the target group is undefended, the sequence branches to 4260 to compute; if I > 0, it branches to 4180 to test A7; if the defense is exhausted, it branches to 4260 to compute. The sequence also steps D, sets I = the number of type D interceptors in the $\vec{\delta}$ strategy, and returns to 4100 to see if I > 0.

```
4180 IF A7<=I THEN 4210
4190 J=I
4200 GOTO 4220
4210 J=A7
4220 A7=A7-J
4230 I=I-J
4240 V=V*(1-P(I5)*D(A(I(25)+D)))↑J
4250 GOTO 4070
```

This sequence sets J = min{A7,I}; reduces both by J; computes a new value of V at 4240; and goes back to 4070 to see if A7 = 0.

```
4260 V=V*(1-P(I5))↑A7
4270 NEXT A
4280 RETURN
```

This sequence computes a new value of V when there is no more defense. When all of the attack weapons, but not necessarily all of the interceptors, have been exhausted, the program returns the computed V to statement 3770.

Some general observations may be useful. SUB 4000 contains the only two statements in the entire program that make direct use of the probability data stored in matrices (P) and (D). This feature gives the operator some latitude to simplify 4240 and 4260 by precomputing some of the factors and storing them in (P) and/or (D) at the time of initialization. Thus, he might save computer processing time by using more storage space and some extra indexing. In fact, the current version of the program contains at statement 540 a precomputation of the complement of PI, but that doesn't require any extra storage. We felt it more desirable at this time to minimize multidimensional storage requirements in (P) at the expense of added processing time. However, if a great many runs were to be made on small-scale problems, it might be judged worthwhile to increase the amount of precomputation in the value function.

Of much greater importance, however, is the flexibility that the operator has to use an entirely different value function, even going so far as to read in all the values as arbitrary input data. In case a different value function is to be used, the operator should provide the following general modifications in the program;

-1 Change the initialization procedures to read in the desired input data and to precompute and store the desired quantities in (P), (D), and any other unused matrix.

-2 Replace SUB 4000 with a subroutine designed to get V for any pair of strategies located by I(41) and I(43) at statement 3760. Be careful to avoid accidently changing any of the variables used elsewhere in the program. In general, the undifferentiated letter variables are available for use, with the exception of G, W, K, and M, which have specified values at the time of access to 4000. Obviously, A7 and I5 are also available. Other variables should be used only after carefully examining the entire program for conflicts.

-3 Make whatever indexing changes are consistent with the new value function; specifically change the setting of I4 at 2030 and its stepping at 3850, if desired.

-4 No other changes need be made as long as the operator is satisfied with the "floating lid" method of generating test strategies. That can be changed too, but this is not the place to discuss how.

G. Print Subroutine

In this version of the program, the print subroutine is addressed only from statement 1870 at the end of a segment. It prints either an attack strategy or a defense strategy, as determined by the current value of S1.

```
5000 DEF FNA(I)=H*Y(I)+X(I)
```

This line defines FNA(I) as a function to be used in combining the (X) and (Y) solution vectors into a single vector by means of H, the value of the parameter.

```
5010 V=0
```

This line initializes the value of the game.

```
5020 FOR W=1 TO K(1)
5030 R=FNA(W)
5040 N=A(L(8)+W)
5050 V=V-R*S(L(3)+N)
5060 PRINT N;-R
5070 NEXT W
```

This sequence computes the marginal value for each of the opposition weapon types currently in play; adds the product of marginal value by number of weapons to V; and prints weapon identifying number and marginal value.

```
5080 I=K(1)
5090 J=K(8)+K(9)*G1
5100 FOR G=1 TO G1
```

This sequence begins a loop, ending at statement 5210, that computes and prints the augmented strategy for each target group.

```
5110 I=I+1
5120 Z=T(G)*FNA(I)
5130 V=V-Z
5140 PRINT ;G;-Z
```

This sequence computes the negative of the per target intercept value, multiplies by T(G) to get the total intercept value for the group, adds this to V, and prints the group number and the group intercept value.

```
5150 FOR N=0 TO N(K(7)+G)
5160 I=I+1
5170 J=J+K(9)
5175 PRINT ;;
5180 FOR L=1 TO L(1)
5185 PRINT A(J+L);
5190 NEXT L
5195 PRINT T(G)*FNA(I)
5200 NEXT N
5210 NEXT G
```

For each elementary strategy on a target group, this sequence prints the number of each type of weapon per target, and the number of targets on which that strategy is used.

```
5220 PRINT "VS="V
5230 RETURN
```

Line 5220 prints VS, the value of the game.

Section VI.B. contains many illustrative printouts.

H. Addition Subroutine

This subroutine is addressed from statement 1560 or statement 1770 when the critical value of the parameter is H3 (implying that a new elementary strategy must be added to the Q-basis). It makes the necessary changes in (A), (R), (S), and the associated indices.

The target group affected is G3, as recorded at 3330, and the numerical elements of the new strategy are on hand in the 31-60 region of (A).

```
6000 G=G3
6010 S1=-S1
6020 GOSUB 8690
6030 GOSUB 8760
6035 GOSUB 8835
6040 S1=-S1
```

This sequence identifies the target group and gets the correct indices from subroutines 8690, 8760, and 8835. For this purpose, S1 is temporarily reversed to orient (K) and (L) properly for adding either a row or a column. The reader will find it helpful to visualize the subroutine as adding a row, and the text will follow this line of thought, with a few identified exceptions.

```
6050 J2=J1+K(9)*(K4+1)
6060 FOR J=J9 TO J2+1 STEP-1
6070 A(J+K(9))=A(J)
6080 NEXT J
6090 FOR N=1 TO L(10)
6100 A(J2+N)=A(30+N)
6110 IF A(J0+N)>A(30+N) THEN 6130
6120 A(J0+N)=A(30+N)
6130 NEXT N
```

This sequence makes all necessary changes in (A) and locates the base strategy row of the G3 + 1 target group in (A). This is the row where the new elementary strategy being added to the G3 group will be stored. The program makes room for the new strategy by shifting upward all higher rows. It stores the new elementary strategy and the weapon sum in the J2 row and changes the test control elements in the J0 row as required by the "floating lid" method of controlling tests. (A different method of controlling tests might call for modifying these operations on the J0 row, or even removing them from the program.)

```
6150 FOR M=((K(6)+G9)-1 TO G(K(6)+G+1) STEP -1
6155 R(M+K(13))=R(M)
6160 S(M+K(13))=S(M)
6170 NEXT M
```

This sequence shifts elements of the α region of (R) and (S) to open the proper row for storing the new strategy.

```
6180 FOR I=G(G9)-1 TO G(G+1) STEP-1
6185 R(I+L4)=R(I)
6190 S(I+L4)=S(I)
6195 NEXT I
6200 IF S1=-1 THEN 6290
```

This sequence shifts elements in the pivot regions of (R) and (S) to open a row for storing the new strategy, performing the shift for those pivot regions pertaining to groups with G > G3. If a true row is to be added, as indicated by S1 = -1, the program branches to 6290 since the required space is now open. This case can be illustrated by Figure 8. When elements 7-12 are shifted to 10-15, spaces 7, 8, and 9 become available for a new row of the first group.

```
6205 I(3)=I(3)+1
6210 I=G(G+1)
6215 J=L4
6220 FOR L=2 TO L4
6222 J=J-1
6225 FOR K=1 TO K4
6230 I=I-1
6240 R(I+J)=R(I)
6250 S(I+J)=S(I)
6260 NEXT K
6280 NEXT L
```

If a true column is to be added, as indicated by S1 = 1, this sequence performs variable shifts to open the proper spaces within the G3 pivot region. As an example, consider the addition of a column to the first group. The elements 7-12 are shifted to 9-14 by the 6180 sequence. Then the 6205 sequence shifts 4-6 to 5-7, so that spaces 4 and 8 become available for the new column. The index I(3) is stepped to reflect the increased row-to-row interval.

```
6290 M5=G(K(6)+G+1)
6300 R(M5)=G(G3)-A(31+L(10))
6310 S(M5)=T(G3)*R(M5)
6330 FOR L=1 TO L(1)
6340 S(M5+L)=T(G3)*(A(30+L)-A(J1+L))
6350 NEXT L
```

This sequence enters the first (or right-side) element of the newly opened row of the α region of (R) in the form $v_{11} - v_{i1}$ without multiplying by T. It then enters the same element of (S). In this version of the program, all elements of (S) contain the scaling factor T(G), but none of the (Q) or (R) elements are scaled. This sequence enters the scaled α differences in (S). These differences are not saved in (R).

```
6360 FOR G=G3+1 TO G9
6365 M(K(7)+G)=M(K(7)+G)+1
6370 G(K(6)+G)=G(K(6)+G)+K(13)
6375 G(G)=G(G)+L4
6380 NEXT G
6390 N(K(7)+G3)=N(K(7)+G3)+1
6400 IF G3>G4 THEN 6730
```

This sequence readjusts all indices affected by the addition of a row to G3. If G3 > G4, the program branches to a RETURN statement, since no further computations are needed on an undefended target group, which has no excess defense strategies and, hence, no pivot region in (S).

```
6405 K4=K4+1
6410 FOR K=K(5)+M(K(7)+G9) TO K0+K4 STEP -1
6420 Q(K+1)=Q(K)
6450 NEXT K
6460 Q(K0+K4)=0
6470 N1=N0
6480 I,I5=I0+I(1)*(K4-1)
6490 J=31+L(10)
```

```
6500 FOR L=1 TO L4
6530 J=J+1
6540 R(I)=A(J)-A(31+L(10))+R(N1)
6545 S(I)=T(G3)*R(I)
6550 I=I+I(3)
6555 N1=N1+L(13)
6560 NEXT L
```

This sequence sets K4 = the new number of rows in the pivot region and readjusts the pivot index (Ø). It enters new values in the open row of the pivot regions of (R) and (S).

```
6570 L6=0
6575 L7=1
6580 L8=I(3)
6585 I2=I5
6590 FOR L1=1 TO L4
6610 IF Ø(L0+L1)=0 THEN 6680
6620 K1=Ø(L0+L1)
6630 Q=T(G3)*R(I2)
6640 S(I2)=S(I2)-Q
6645 M3=M0+K(13)*(K1-1)
6650 I3=I0+I(1)*(K1-1)
6660 GOSUB 8532
6670 GOTO 6690
6680 L6=L1
6690 I2=I2+I(3)
6695 NEXT L1
```

This sequence operates on the new row of (S) to reflect the effect of pre-existing pivots in the other rows of (S). It initializes L6, a variable used to record any column without a pivot that may be found during the ensuing process; presets L7 and L8, interval step indicators needed in SUB 8532; and initializes I2 as the first element of the new row in the pivot region. It loops on columns. If there is no pivot in the L1 column, the program branches to 6680, records L6 = L1, steps I2, and goes to the next column. If there is a pivot in the L1 column, the program records its row as K1, presets Q, a multiplier used in SUB 8532, replaces

S(I2) by S(I2) - Q, locates the first elements of the K1 row in the α region and the pivot region, goes to SUB 8532, and then goes to the next column.

```
6700 IF L6=0 THEN 6730
6705 K=K4
6710 I=I5+I(3)*(L6-1)
6720 IF ABS(S(I))>.1 THEN 8110
6730 RETURN
```

If L6 = 0, all columns have pre-existing pivots, and the program branches to 6730. Otherwise, it identifies the potential pivot row as K, the column being L6 > 0, and sets I as locator of the potential pivot element. If S(I) ≠ 0, the program branches to 8110 to initiate the pivot; otherwise, it returns to the control section.

I. Deletion Subroutine

This subroutine is addressed from statement 1580 or 1800 when the critical value of the parameter is H2 (implying that an elementary strategy must be deleted from the hypothetical Q-basis). It makes the necessary changes in (A), (Q), (R), (S), and the associated indices.

The target group affected is G2, and the elementary strategy within this group is M2, as recorded at 3070 and 3080, respectively. The deletion algorithms depend on the pivot status of this elementary strategy.

```
7000 G=G2
7010 GOSUB 8760
7015 GOSUB 8835
7020 L6=0
7030 J2=J1+K(9)*M2
7040 IF M2>0 THEN 7580
```

This sequence identifies the target group and gets the correct indices from subroutines 8760 and 8835. Since the (K) and

(L) indices are already correctly oriented, there is no need to go to SUB 8690. The sequence initializes L6, a locator that will be used to record the pivot column if the row being deleted is a pivot row and computes the location in (A) of the row being deleted. If M2 > 0, the row being deleted appears explicitly in (S), and the program branches to 7580.

```
7050 I5=I0
7052 L7=1
7055 L8=I(3)
7060 FOR K=1 TO K4
7070 J2=J2+K(9)
7090 GOSUB 8570
7092 IF C(KC+K)>0 THEN 7100
7095 Q=Q-1
7100 IF ABS(Q)>.0001 THEN 7120
7105 I5=I5+I(1)
7110 NEXT K
```

If M2 = 0, the base row must be replaced, and the sequence through line 7570 controls the replacement algorithms. The 7050 sequence selects one of the rows in (S) to be the new base by looping on the rows of the pivot region, going to SUB 8570 to compute Q, and exiting from the loop when Q ≠ 0.

```
7120 K1,M2=K
7130 I1,I3=I5
7135 M1,M3,M5=M0+K(13)*(K1-1)
```

This sequence sets locators needed later; identifies the new base row, which will be deleted from (S), as K1, M2; locates its first element in the pivot region as I1, I3; and locates its first element in the α region as M1, M3, M5.

```
7140 I,I5=I0
7145 M=M0
7150 FOR K=1 TO K4
7155 IF K=K1 THEN 7220
7157 R(M)=R(M)-R(M1)
7160 I1=I3
7170 FOR L=1 TO L4
7180 R(I)=R(I)-R(I1)
7190 I=I+I(3)
7200 I1=I1+I(3)
7210 NEXT L
7220 I,I5=I5+I(1)
7225 M=M+K(13)
7230 NEXT K
7235 Q(2)=Q(2)-R(M1)
7240 I1=I3
7245 N=N0
7250 FOR L=1 TO L4
7255 R(N)=R(N)-R(I1)
7260 I1=I1+I(3)
7265 N=N+L(13)
7267 NEXT L
```

This sequence recomputes the (R) matrix to reflect the change of base, and recomputes the base value stored in (Q).

```
7270 FOR N=1 TO L(10)
7280 A(J1+N)=A(J2+N)
7300 NEXT N
```

This sequence substitutes the new base row in (A).

```
7310 [illegible]
7315 I5=I3
7320 IF S(K6+K1)=0 THEN 7370
7330 L1,L6=S(K6+K1)
7340 I6=I3+I(3)*(L1-1)
7350 R(I6)=1
7360 GOSUB 8532
```

This sequence begins recomputation of (S) to reflect the change of base. It performs the computation on the row of (S) identified as the new base row. If this is a pivot row, the sequence records the column number of the pivot element as L1, L6 for future use. Later this row will be deleted from (S), but it must first be modified as part of the algorithm for modifying the other rows.

```
7370 I5=N0
7372 I2=I5+L(13)*(L1-1)
7375 M5=I(36)
7377 L7=L(12)
7378 L8=L(13)
7380 FOR K=0 TO K(1)
7390 GOSUB 8570
7395 IF K>0 THEN 7410
7400 G=G-1
7410 IF G(K0+K1)=0 THEN 7430
7420 S(I2)=0
7430 GOSUB 8532
7435 I5=I5+1
7440 I2=I2+1
7445 M5=M5+K(12)
7450 NEXT K
```

This sequence performs the recomputation for the rows of the common and δ regions.

```
7455 I5=I0
7460 I2=I5+I(3)*(L1-1)
7462 L7=1
7463 L8=I(3)
7465 M5=M0
7467 FOR K=1 TO K4
7480 IF K=K1 THEN 7525
7490 GOSUB 8570
7492 IF G(K0+K)>0 THEN 7500
7495 G=G-1
7500 IF G(K0+K)=0 THEN 7520
7510 S(I2)=0
7520 GOSUB 8532
7525 I5=I5+I(1)
7530 I2=I2+I(1)
7535 M5=M5+K(13)
7537 NEXT K
```

This sequence performs the recomputation for the rows of the α and pivot regions.

```
7540  IF P(K0+K1)=0 THEN 7820
7550  P(K0+K1),P(L0+L1)=0
7560  P8=P8-1
7570  GOTO 7820
```

If the new base row is not a pivot row, the program branches at once to 7820. Otherwise, it sets the pivot indices to 0 and reduces the pivot counter P8, since the row is going to be deleted from (S).

```
7600  IF P(K0+K1)=0 THEN 7820
7610  L6=P(K0+K1)
7620  I9=I0+I(1)*(M2-1)+I(3)*(L6-1)
7630  S3=-1
7640  IF ABS(S(I9))>.001 THEN 7780
```

This sequence is reached only from 7040. If the row to be deleted is not a pivot row, the program branches to 7820. Otherwise, it begins recomputing the (S) matrix prior to de[illegible]ng a pivot row, using an algorithm best described as an "unpivot." The sequence records the column number of the pivot element as L6, locates the pre-existing pivot element, sets the sensor S3 = -1 to indicate an unpivot, and branches to 7780 if the pivot element S(I9) ≠ 0.

```
7650  I3=I0
7660  FOR K1=1 TO K2
7670  IF P(K0+K1)=0 THEN 7750
7680  IF K1=M2 THEN 7750
7690  L1=P(K0+K1)
7700  I6=I3+I(2)*(L1-1)
7710  IF ABS(S(I6))<.001 THEN 7750
7720  GOSUB 8180
7730  P(K0+K1),P(L0+L1)=0
7740  IF ABS(S(I9))>.001 THEN 7780
7750  I3=I3+I(1)
7755  NEXT K1
7760  PRINT "UNPIVOT ERROR S(I9)="S(I9)
7770  STOP
```

If S(I9) = 0, this sequence invokes the unpivoting procedure on one or more other pivots of the group in order to make the desired pivot element S(I9) ≠ 0, tests at 7740 each time, and branches to 7780 whenever the condition is satisfied. If the condition cannot be satisfied by removing all other pivots in the group, there is an error printout and programed STOP at 7770. The error printout has never occurred in any run of the debugged program. However, the procedure is a messy one, and a better recursion algorithm is needed.

```
7780 K1=K2
7790 L1=L2
7800 GOSUB 8160
7810 [illegible](K0+K1),[illegible](L0+L1)=0
```

If S(I9) ≠ 0, this sequence denotes the pivot row and column by K1 and L1, goes to subroutine 8160 with S3 = -1 for the unpivot computations, and sets the pivot index at 0.

```
7820 FOR J=J2+1 TO J9
7830 A(J)=A(J+K(9))
7840 NEXT J
7850 I2,I5=I0+I(1)*(M2-1)
7860 FOR M=M0+K(12)+(M2-1) TO ((K(0)+G9)-1
7865 F(M)=F(M+K(13))
7870 S(M)=S(M+K(13))
7880 NEXT M
7885 IF S3=1 THEN 7915
7890 I(3)=I(3)-1
7892 J=1
7894 FOR L=2 TO L4
7896 FOR K=2 TO K4
7898 F(I2)=F(I2+J)
7900 S(I2)=S(I2+J)
7902 I2=I2+1
7904 NEXT K
7906 J=J+1
7910 NEXT L
7915 FOR I=I2 TO G(69)-1
7920 F(I)=F(I+L4)
7925 S(I)=S(I+L4)
7930 NEXT I
```

This sequence deletes the M2 row from (A), (R), and (S), by shifting down all the higher rows. If S1 = -1, the sequence 7890-7910 is needed to close the separated spaces within the G2 pivot region of (R) and (S). It is the counterpart of the sequence 6205-6280 for opening spaces.

```
7935 FOR K=M2 TO K4-1
7940 S(K0+K)=S(K0+K+1)
7945 IF S(K0+K)=0 THEN 7955
7950 G(L0+G(K0+K))=K
7955 NEXT K
7960 FOR K=K0+K4 TO K(5)+M(K(7)+G9)
7965 S(K)=S(K+1)
7970 NEXT K
```

This sequence shifts the elements of the pivot index to conform to the new structure of (S).

```
7980 K4=K4-1
7990 FOR G=G2+1 TO G9
7995 M(K(7)+G)=M(K(7)+G)-1
8000 G(K(6)+G)=G(K(6)+G)-K(13)
8005 C(G)=C(G)-L4
8010 NEXT G
8020 N(K(7)+G2)=N(K(7)+G2)-1
```

This sequence changes other indexes to conform.

```
8030 IF L6=0 THEN 8100
8040 I=I0+I(3)*(L6-1)
8050 FOR K=1 TO K4
8070 IF S(K0+K)>0 THEN 8090
8080 IF ABS(S(I))>.1 THEN 8110
8090 I=I+I(1)
8095 NEXT K
8100 RETURN
```

If L6 = 0, the program branches to 8100, since no pivots have been removed. Otherwise, a pivot has been removed by deleting a row, so the other elements of the old pivot column are tested for potential pivots, and if one is found, the program branches to 8110 to carry out the pivot operation. Otherwise, the subroutine is finished.

```
8110 K1,U(L0+L6)=K
8120 L1,U(K0+K1)=L6
8130 S3=1
8140 GOSUB 8160
8150 RETURN
```

This sequence identifies the pivot row and column as K1 and L1, respectively, records the pivot indices, sets the sensor S3 at +1 to indicate a pivot (as opposed to an "unpivot," which would be indicated by -1), goes to SUB 8160 to make the pivot, and returns to the control program.

J. <u>Auxiliary Subroutines</u>

```
8160 I3=I0+I(1)*(K1-1)
8165 M3=M0+M(I3)*(K1-1)
8170 I6=I3+I(3)*(L1-1)
8180 P=S3*S(I6)
8190 S(I6)=-S3
8200 Q=1-1/P
8210 I5=I3
8215 M5=M3
8217 L7=1
8218 L8=I(3)
8220 GOSUB 8532
```

This subroutine (through 8390) controls the pivot operation, addressing SUB 8525 for the actual computation of new values. This sequence locates the first element of the pivot row in the pivot region and in the α region, and locates the pivot element. Then it prepares for computation of the pivot row, sets the multiplier Q so that SUB 8532 will have the effect of dividing each element of the pivot row by S3 times the pivot element, resets the pivot element so that SUB 8532 will have the effect of replacing the original pivot element by its negative reciprocal, initializes I5 and M5 at the first elements of the row, sets the stepping indicators L7 and L8, and goes to SUB 8532 to compute.

```
8230 I5=NC
8235 M5=I(36)
8240 I2=I5+L(13)*(L1-1)
8245 L7=L(12)
8246 L8=L(13)
8250 FOR K=0 TO K(1)
8255 GOSUB 8525
8260 I5=I5+1
8270 I2=I2+1
8280 M5=M5+K(12)
8290 NEXT K
```

This sequence prepares for computation of the rows in the common and δ regions, initializes I5, M5, L7, and L8, initializes I2 at the pivot column element of the first row, and loops on the rows, going to SUB 8525 to compute.

```
8300 I5=I0
8301 I2=I5+I(3)*(L1-1)
8302 L7=1
8303 L8=I(3)
8307 M5=M0
8320 FOR K=1 TO K4
8322 IF K=K1 THEN 8330
8325 GOSUB 8525
8330 I5=I5+I(1)
8335 M5=M5+K(13)
8340 I2=I2+I(1)
8370 NEXT K
8380 P8=P8+S3
8390 RETURN
```

This sequence prepares for computation of the rows in the α and pivot regions, except the pivot row, which has already been computed. It maintains the count of the current number of explicit pivots in (S) by replacing P8 by P8 + S3.

```
8525 Q=S3*S(I2)
8530 S(I2)=0
8532 I=M5
8536 FOR L=0 TO L(1)
8540 S(I)=S(I)-S(M3+L)*Q
8541 I=I+L7
8542 NEXT L
8546 I=I5
8548 I1=I3
8550 FOR L=1 TO L4
8552 S(I)=S(I)-S(I1)*Q
8554 I=I+L8
8556 I1=I1+I(3)
8558 NEXT L
8560 RETURN
```

This computational subroutine is addressed from a number of places in the program. It loops simultaneously across two parallel rows in (S), subtracting from the elements of one row the product of a multiplier Q and the elements of the other row. In some instances, the first two statements of the subroutine are by-passed. When used, they initialize the multiplier and the pivot-column element S(I2) of the row being modified, so that statement 8552 will have the effect of dividing this element by S3 times the original pivot element.

```
8570 Q=0
8580 I=I5
8590 FOR L=1 TO L4
8610 IF L(L6+L)=0 THEN 8630
8620 Q=Q+S(I)
8630 I=I+L8
8640 NEXT L
8680 RETURN
```

This subroutine is addressed for a specified row of either the δ region or the pivot region of (S). It sums the elements in pivot columns of that row. It is used only when replacing a base row.

```
8690 M=0
8700 FOR N=1 TO 13
8710 M=M+3
8720 K(N)=I(M-S1)
8730 L(N)=I(M+S1)
8740 NEXT N
8750 RETURN
```

This subroutine reads one wing of (I) into (K) and the other into (L). When the sign of S1 is reversed, the wings are interchanged and the problem is effectively transposed.

```
8760 K0=K(5)+M(K(7)+G)
8775 K4=N(K(7)+G)
8790 L0=L(5)+M(L(7)+G)
8805 L4=N(L(7)+G)
8820 I0=G(G)
8825 M0=G(K(6)+G)
8830 N0=G(L(6)+G)
8831 I(2+S1)=1
8832 I(2-S1)=N(G9+G)
8833 RETURN
```

This subroutine names as simple variables a number of frequently used indexes associated with a target group G. K0, L0 are row, column locators for (Ø). K4, L4 are the number of rows, columns in the pivot region of (S). I0 is the location of the first element in the pivot region of (S). M0 is the location of the first element in the α region of (S). N0 is the location of the first element in the δ region of (S). I(1) is the interval between elements of successive rows in the same column of the pivot region. I(3) is the interval between elements of successive columns in the same row of the pivot region.

```
8835 J0=K(8)+K(9)*G
8840 J1=K(8)+K(9)*(G1+M(K(7)+G)+G)
8850 J9=K(8)+K(9)*(G1+M(K(7)+G9)+G9)
8860 RETURN
```

This subroutine computes several indexes to (A). J0 locates the test control row for group G. J1 locates the base strategy row for group G. J9 locates the base strategy row for the hypothetical group G9.

K. <u>Strategy Subroutine</u>

This subroutine is regularly entered from statements 1520 and 1720 in the control program. There is also a special short-cut entry from statement 1100, used to cause a strategy printout without changing resources.

The first portion of the subroutine sets up the third stage matrix (U) and inverts it into (W). The remainder carries the back solutions through the third, second, and first stages. The end product is two vectors, (X) and (Y), representing, respectively, the first and second columns of the solution strategy expressed in terms of the particular parameter being used at the moment. (X) and (Y) are normally not combined into a single solution strategy vector except during printout in SUB 5000.

The subroutine serves eight cases, corresponding to the combinations of values that S1, S2, and P2 may have in the control program.

```
9000 I9=M1-K8
9005 MAT U=ZER(I9,I9)
9010 MAT W=ZER(I9,I9)
9015 MAT Z=ZER(I9,I9)
```

This sequence zeros (U), (W), and (Z) at the greater dimension of (S) less the number of pivots in (S).

```
9020 M=I(36)
9025 FOR K=1 TO I(2)
9030 M=M+I(35)
9035 FOR L=1 TO I(4)
9040 U(K,L)=S(M+L)
9045 NEXT L
9050 NEXT K
```

This sequence enters the common region of (S) into (U). The use of (I) indices means that (U) is set up without transposition.

```
9070 K3=I(2)
9075 L3=I(4)
9080 FOR G=1 TO G1
9085 I,I3=G(G)
9090 M,M3=G(G9+G)
9095 N,N3=G(G8+G)
9100 K1,K2=K3
9105 L1,L2=L3
9110 K6=K(G)
9115 L6=I(16)+G(G9+G)
9117 L4=G(G9+G)
9120 FOR K=1 TO N(G)
9125 IF G(K6+K)>0 THEN 9160
9140 K3=K3+1
9145 FOR L=1 TO I(4)
9150 U(K3,L)=S(M3+L)
9155 NEXT L
9160 M3=M3+I(35)
9161 NEXT K
9162 FOR L=1 TO L4
9170 IF G(L6+L)>0 THEN 9196
9172 L3=L3+1
9174 FOR K=1 TO I(2)
9176 U(K,L3)=S(I+K)
9180 NEXT K
9182 K1=K2
9184 FOR K=1 TO N(G)
9186 IF G(K6+K)>0 THEN 9194
9190 K1=K1+1
9192 U(K1,L3)=S(I)
9194 I=I+L4
9195 NEXT K
9196 I,I3=I3+1
9197 N,N3=N3+I(40)
9198 NEXT L
9200 NEXT G
```

This sequence enters into (U) the other elements of (S) at the intersection of nonpivot rows and nonpivot columns. The pivot index (Ø) controls the selection. Elements of the α region are entered by statement 9150, of the δ region by 9178, and of the pivot region by 9192.

```
[illegible] IF [illegible]=1 THEN [illegible]
[illegible] IF [illegible]=-1 THEN [illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

If S2 = 1, (U) is a nonsingular matrix and may be inverted. If S2 = -1, the last row or column of (U) consists of zeros, and the deficiency is made up by entering P2 in the R5 element of the last row or column.

```
[illegible]
[illegible] IF [illegible] THEN [illegible]
[illegible]
[illegible]
[illegible]
```

The solutions are sensitive to the orientation of (S) and (W), so subroutine 8690 is called to provide (K) and (L) indices, and (Z) is set as either (W) or the transpose of (W).

```
[illegible]
[illegible] IF [illegible] THEN [illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

```
9415 L1=L(1)
9417 L2=L(5)
9418 M=L(L(2)+1)
9420 FOR L=1 TO Y(L(2)+19)
9430 IF Z(L2+L)>0 THEN 9446
9435 L1=L1+1
9440 FOR K=1 TO 19
9442 C(K)=C(K)+Z(M)*S(L1,K)
9444 NEXT K
9446 M=M+L(13)
9448 NEXT L
```

This sequence computes the third-stage solution from (Z) and the right-side elements of (S), storing the solution in (C) and (F).

```
9450 FOR K=1 TO K(1)
9460 X(K)=C(K)
9462 Y(K)=F(K)
9464 NEXT K
```

This sequence enters the marginal returns from (C) and (F) directly into (X) and (Y).

```
[illegible] J1=L(1)+L(4)*[illegible]
[illegible] I1=K(1)+1
[illegible] K2=K(1)
[illegible] FOR G=1 TO [illegible]
[illegible]
```

This sequence begins the back solution by a loop on G extending to line 9900. For G = 1, it initializes J1 to locate the base strategy in (A) and I1 to locate the intercept elements in (X) and (Y). It then initializes K2 to locate the last elements of (C) and (F) already transferred to (X) and (Y). For each group in turn, it goes to SUB 8760 to get specific indexes.

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

This sequence begins the back solution for elements of (X) and (Y) corresponding to excess strategies. The loop on I extends to line 9700. If there is a pivot in row K1 of (S), the program branches to 9610; otherwise, it continues.

```
[illegible]
[illegible] (I)=[illegible]
[illegible] Y(I)=[illegible]
[illegible]
```

If the K1 row of (S) is not a pivot row, the corresponding solution elements (X) and (Y) are entered directly from (C) and (F).

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

If the K1 row of (S) is a pivot row, the corresponding elements of (X) and (Y) must be computed from (C), (F), and all those elements of (S) in the column of that pivot which are not themselves in any pivot row. The computation runs first over the marginal-value elements of (C) and (F).

```
    [illegible] K5=K[illegible]
    [illegible] FOR K=1 TO K4
    [illegible] IF [illegible](K(I+K)>0 THEN [illegible]
    [illegible] K5=K5+1
    [illegible] X(I)=X(I)-C(K5)*S(I[illegible])
    [illegible] Y(I)=Y(I)-F(K5)*S(I[illegible])
    [illegible] I[illegible]=I[illegible]+I[illegible])
    9695 NEXT K
    [illegible] NEXT I
```

The computation is completed by running over other elements of (C) and (F) pertaining to target group G.

```
    [illegible] X(I1)=-[illegible]
    [illegible] J=I1+1
    [illegible] X(J)=1
    [illegible] Y(I1),Y(J)=0
```

The back solution for intercept value, X(I1) and Y(I1), and for base strategy, X(J) and Y(J), begins with their initialization.

```
    [illegible] I5=[illegible]
    [illegible] FOR I=I1+[illegible] TO I1+1+K4
    [illegible] X(J)=X(J)-X(I)
    [illegible] Y(J)=Y(J)-Y(I)
    [illegible] X(I1)=X(I1)+X(I)*R(I5)
    [illegible] Y(I1)=Y(I1)+Y(I)*R(I5)
    [illegible] I5=I5+A([illegible])
    [illegible] NEXT I
    [illegible] FOR I=1 TO [illegible]
    [illegible] X(I1)=X(I1)-X(I)*A([illegible]+I)
    [illegible] Y(I1)=Y(I1)-Y(I)*A(J1+I)
    [illegible] NEXT I
```

This sequence completes the back solution for intercept and base strategy, using previously computed elements of (X) and (Y) and stored elements of (R) and (A).

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

This sequence steps indexes for the next target group. After completing (X) and (Y), it returns to the control program.

The END statement is not a part of the operating program but is required by the computer system.

V. PATH87A COMPUTER PROGRAM

One-sided optimization problems can be run on the PATH87 program by specifying no defended target groups and no defense weapon types, i.e., G4, D1 = 0. However, much of the capability of the program is not needed for these simpler problems and can be cut out to save running time.

The PATH87A program is designed for one-sided optimizations. Being derived from PATH87, it has a similar general structure but is less than half the length. Excess capabilities have been cut out and many procedures simplified.

A. Simplifying Ideas

By definition, the defense solution strategy is a null strategy, which need not be computed or stored. Since there are no excess elementary defense strategies, the pivot regions of (R) and (S) are nonexistent, and the second-stage solution procedures can be eliminated, along with the pivot index (Ø). If the components of the attack solution strategy are redefined as numbers of targets, rather than as fractions of a group of targets, the scaling factor T(G) can be eliminated from (S), which reduces round-off error as well as computational time. The one-sided nature of the problem results in most of the subroutines having to work only one way rather than two. Consequently, the reversible indexing system is not needed, and we can eliminate (K) and (L).

B. Program Listing

The following listing has explanatory comments added where appropriate:

```
1C7487A

50 DATA 5,3
60 DATA 1,10,15,8,10,6,20,5,50,2
70 DATA .8,.5,.5,.7,.6,.3,.7,.6,.4,.9,.5,.3,.4,.2,.5
```

Only two dimensions, G1 and A1, need to be specified. Line 80 is omitted.

```
100 DIM G(100),T(100),V(100),M(202),N(202),C(101),J(202)
105 DIM C(15),F(15),S(15),U(15,15),V(15,15)
110 DIM I(40)
115 DIM X(250),Y(250)
120 DIM S(900)
125 DIM A(4000),F(1500)
```

This sequence omits matrices (D), (K), (L), (Ø), (R), and (Z).

(G) is reduced in size since only the α region of (S) needs indexes.

(Q) is enlarged to store the value associated with every elementary attack strategy, not just the base strategies.

(M) is used to locate the base value for group G in (Q).

(N) is reduced to store only the number of rows for each target group in (S).

(J) is added as a locator for the attack region of (A). It is divided into two regions, J(1) to J(G9) locating base-strategy rows and J(G9+1) to J(2*G9) locating control rows for each group. The defense region of (A) is not used.

```
130 READ G1,A1
135 PRINT "G1,A1="G1;A1
140 G9,I(17),I(22)=G1+1
145 I(7),I(10)=A1
155 I(37)=1
160 I(16)=30
```

```
165 I(35),I(36)=A1+1
180 I(23)=60
190 I(27),I(29),I(31)=1
195 I(26)=A1+I(27)
205 I(32),G(1)=1+I(35)
206 I(36)=1
207 M(1)=1
210 J(G9+1)=I(23)+I(26)
215 J(1)=J(G9+1)+I(1)+I(26)
```

This sequence reads dimensions of the problem and initializes indices. In actuality, the program could be improved by replacing the (I) elements by simple variables throughout.

```
300 PRINT "G,V="
310 FOR G=1 TO G1
320 READ T(G),V(G)
330 PRINT T(G);V(G)
340 U(G)=V(G)
350 V1=V1+V(G)
360 G(G+1)=G(1)
370 M(G+1)=G+1
380 J(G9+G+1)=J(G9+G)+I(26)
385 J(G+1)=J(G)+I(26)
390 NEXT G
```

This sequence reads target data and initializes the (G), (M), and (J) indices.

```
400 PRINT "PKT="
410 I=0
420 FOR G=1 TO G1
430 FOR A=1 TO A1
440 I=I+1
450 READ P
460 PRINT P;
465 P(I)=1-P
470 NEXT A
480 PRINT
490 NEXT G
```

This sequence reads probabilities of kill and stores probabilities of survival in (P).

```
1000 PRINT "P1,P3?"
1010 INPUT P1,P3
1020 IF P1<>0 THEN 1040
1030 STOP
```

Meaningful inputs for P1 are positive integers to denote attack weapon types, 0 to terminate the run, and -1 to get a printout of marginal values. Their use is illustrated in Section VI.A.

```
1040 S1=SGN(P1)
1050 IF S1=-1 THEN 1100
1070 P2=SGN(P3-$(P1))
1075 V2=V1*(1-P2)
1080 P4=ABS(P3-$(P1))
1090 IF P4>0 THEN 1130
1100 GOSUB 9280
1110 GOTO 1860
1130 FOR R5=1 TO A2
1140 IF A(I(23)+R5)=P1 THEN 1700
1150 IF A(I(23)+R5)>P1 THEN 1180
1160 NEXT R5
1170 R5=A2+1
1180 FOR R=A2 TO R5 STEP-1
1254 I=1+R
1256 FOR K=0 TO A2
1260 S(I+1)=S(I)
1262 S(I)=0
1263 I=I+I(38)
1264 NEXT K
1270 NEXT R
1280 FOR J=I(23) TO J(69)-I(26) STEP I(26)
1290 FOR L=J+I(31) TO J+R5 STEP -1
1300 A(L+1)=A(L)
1310 NEXT L
1320 A(J+R5)=0
1330 NEXT J
1340 A(I(23)+R5)=P1
1350 A2=A2+1
1380 I(31)=I(31)+1
```

This sequence initializes a path segment in the same general fashion as in PATH87, but with many simplifications. The number of attack weapon types actually in play is denoted by

A2 rather than by I(4).

```
1500 J1=-1
1520 GOSUB 9000
1530 H3=V2
1540 GOSUB 2000
1560 GOSUB 7000
1700 J1=1
1720 GOSUB 9000
1730 H2=14
1740 GOSUB 2000
1745 IF H2=14 THEN 1950
1800 GOSUB 7000
1810 GOTO 1500
```

The 1500 and 1700 sequences are much simplified.

```
1920 S(I1)=13
1950 S(I(36)+25)=S(I(36)+H5)+12+14
1960 H=14
1970 GOSUB 5000
1980 GOTO 1000
```

This sequence terminates the path segment.

```
2000 IF J1=1 THEN 2500
2020 I4=0
2040 FOR I=1 TO 11
2050 J0=J(I4+1)
2060 J1=J(I)
2070 X1=-S(N(I))
2080 Y1=0
2090 FOR I=1 TO 42
2100 X1=X1-X(I)*A(J1+I)
2105 Y1=Y1-Y(I)*A(J1+I)
2110 NEXT I
```

This sequence begins the test control section for a defense strategy by initializing X1 and Y1. The procedure permits SUB 9000 to be greatly simplified.

```
2120 GOSUB 3200
2130 GOTO 2190
2140 GOSUB 3500
2150 GOTO 2170
2160 K=1
2170 IF A(K)>A(J0+K) THEN 2230
2180 GOSUB 3600
2190 IF Y1>-.001 THEN 2220
2200 GOSUB 4000
2210 GOSUB 3300
2220 IF A(I(31))<=A(J0+I(31)) THEN 2160
2230 IF K<A2 THEN 2140
2245 I4=I4+A1
2250 NEXT G
2260 RETURN
```

This sequence completes the test control section for a defense strategy with essentially the same flow illustrated in Figure 11. Three exceptions should be noted. SUB 3000 is omitted since the defense strategy is unchangeably a null strategy. SUB 4000 is addressed directly instead of through SUB 3700. SUB 3820 is replaced by a single statement stepping I4.

```
2500 FOR G=1 TO G1
2510 GOSUB 3000
2670 NEXT G
2680 RETURN
```

The test control section for an attack strategy calls on SUB 3000 only.

```
3000 FOR I=M(G) TO M(G)+N(G)
3030 IF Y(I)>-.0001 THEN 3090
3040 H=-X(I)/Y(I)
3050 IF H2<H THEN 3090
3060 H2=H
3070 G2=G
3080 M2=I-M(G)
3090 NEXT I
3100 RETURN
```

```
3200 FOR K=1 TO I(31)
3210 A(K)=0
3220 NEXT K
3230 RETURN
3300 H=-(X1+X2)/Y1
3310 IF H3<H THEN 3410
3320 H3=H
3330 G3=0
3370 FOR K=1 TO I(31)
3380 A(K+30)=A(K)
3390 NEXT K
3400 A(30+I(31))=H3
3410 RETURN
3500 X1=X1-A(W)*X(W)
3510 Y1=Y1-A(W)*Y(W)
3520 A(I(31))=A(I(31))-A(W)
3530 A(W)=0
3540 W=W+1
3550 RETURN
3600 A(W)=A(W)+1
3610 A(I(31))=A(I(31))+1
3620 X1=X1+X(W)
3630 Y1=Y1+Y(W)
3640 RETURN
```

The 3000-series subroutines are only five in number, and these are simplified.

```
4000 X2=V(0)
4020 FOR A=1 TO 22
4060 A7=A(A)
4070 IF A7=0 THEN 4270
4080 I5=14+A(I(23)+A)
4260 X2=X2+V(I5)*A7
4270 NEXT A
4280 RETURN
```

The simplified value function computes X2 directly. The variable Y2 is omitted since it would always be zero.

```
5000 DEF FNA(I)=H*Y(I)+X(I)
5010 V=0
5015 IF S1=1 THEN 5080
5020 FOR K=1 TO A2
5060 PRINT A(I(23)+K);-X(K)
5070 NEXT K
5075 RETURN
```

The print subroutine begins with one branch printing only the part of a defense strategy that specifies marginal values for the attack weapons.

```
5080 I=0
5090 J=J(G9+G1)
5100 FOR G=1 TO G1
5140 PRINT ;G
5150 FOR N=0 TO N(G)
5160 I=I+1
5170 J=J+I(26)
5175 PRINT I;
5180 FOR L=1 TO A2
5185 PRINT A(J+L);
5190 NEXT L
5192 T=FNA(I)
5194 PRINT T;
5196 IF T=0 THEN 5204
5198 Z=L(I)*T
5200 PRINT Z;
5202 V=V+Z
5204 PRINT
5206 NEXT N
5210 NEXT G
5220 PRINT "VS="V
5230 RETURN
```

The other branch prints an attack strategy with no marginal values or intercept values. VS is computed as the sum of the products (numbers of targets times expected surviving value per target). Illustrative printouts are in Section VI.A.

```
6000 J1=J(L3)
6010 J0=J(L9+L3)
6020 J2=J(L3+1)
6030 M3=M(L3+1)
6040 FOR J=J(L9) TO J2+1 STEP -1
6070 A(J+I(26))=A(J)
6080 NEXT J
6090 FOR N=1 TO I(41)
6100 A(J2+N)=A(J0+N)
6110 IF A(J0+N)>A(J0+N) THEN 6130
6120 A(J0+N)=A(30+N)
6130 NEXT N
6150 FOR M=M(L9)-1 TO M[illegible]-1
6160 G(M+1)=G(M)
6170 NEXT M
6180 FOR I=G(L9)-1 TO G(L3+1) STEP -1
6190 G(I+I(3[illegible]))=G(I)
6195 NEXT I
6290 I=G(L3+1)
6300 G(M3)=A(31+I(31))
6310 G(1)=G(M(L9))-G(M3)
6330 FOR L=1 TO 62
6340 G(I+L)=A(30+L)-A(J1+L)
6350 NEXT L
6360 FOR L=L3+1 TO [illegible]
6365 M(L)=M(L)+1
6367 J(L)=J(L)+I(26)
6370 G(L)=G(L)+I(38)
6380 NEXT L
6390 N(L3)=N(L3)+1
6730 RETURN
7000 I1=G(L2)
7010 J1=J(L2)
7020 M1=M(L2)
7040 IF M2>0 THEN 7[illegible]0
7110 M2=1
7120 J2=J1+I(26)
7140 G(M1)=G(M1+M2)
7170 FOR L=0 TO 62
7180 S=G(I1+L)
7185 S5=S*T(L2)
7190 G(I1+L)=G(I1+L)-S5
7200 I=I1+L
7210 FOR K=2 TO N(L2)
7220 I=I+I(38)
7230 G(I)=G(I)-S
7240 NEXT K
7250 NEXT L
```

```
7280 FOR N=1 TO I(31)
7290 A(J1+N)=A(J2+N)
7300 NEXT N
7820 FOR J=J1+M2*I(26)+1 TO J(G9)
7830 A(J)=A(J+I(26))
7840 NEXT J
7842 FOR M=M1+M2 TO M(G9)
7844 U(M)=U(M+1)
7846 NEXT M
7850 FOR I=I1+(M2-1)*I(38) TO G(G9)-1
7860 S(I)=S(I+I(38))
7870 NEXT I
7990 FOR G=G2+1 TO G9
7995 M(G)=M(G)-1
7997 J(G)=J(G)-I(26)
8000 G(G)=G(G)-I(38)
8010 NEXT G
8020 N(G2)=N(G2)-1
8150 RETURN
```

The basis subroutines, 6000 and 7000, realize the greatest saving of statement lines by the elimination of the second solution stage and all the complicated algorithms associated with it.

```
9000 MAT U=ZER(A2,A2)
9010 MAT W=ZER(A2,A2)
9030 M=I(36)
9040 FOR K=1 TO A2
9050 M=M+I(38)
9060 FOR L=1 TO A2
9070 U(K,L)=S(M+L)
9080 NEXT L
9090 NEXT K
9220 IF S1=1 THEN 9270
9260 U(A2,K5)=-F2
9270 MAT W=INV(U)
9280 IF S1=-1 THEN 9411
```

The matrix (U) is set up from only one region of (S) instead of four. If S1 = -1, the program goes to 9411 to compute a defense strategy. Otherwise, it continues at 9330.

```
9380 FOR K=1 TO 62
9390 F(K)=F2*V(15,K)
9400 C(K)=0
9402 M=I(36)
9404 FOR L=1 TO 62
9407 C(K)=C(K)+S(M+L)*Z(L,K)
9408 NEXT L
9409 NEXT K
9410 GOTO 9490
```

This sequence begins to compute an attack strategy by storing the third-stage solutions in (C) and (F). The program continues at 9490.

```
9411 FOR K=1 TO 62
9413 Y(K)=Z(K,62)
9414 X(K)=0
9415 NEXT K
9418 M=I(36)
9420 FOR L=1 TO 62
9425 M=M+I(38)
9440 FOR K=1 TO 62
9442 X(K)=X(K)+S(M)*Z(K,L)
9444 NEXT K
9446 NEXT L
9485 RETURN
```

This sequence computes a defense strategy, marginal values only, storing the third-stage solutions directly in (X) and (Y); then it returns to the control program.

```
9490 K2=0
9500 FOR G=1 TO G1
9510 J=M(G)
9520 X(J)=I(G)
9530 Y(J)=0
9540 FOR I=J+1 TO J+N(G)
9570 K2= K2+1
9580 X(I)=C(K2)
9590 Y(I)=F(K2)
9760 X(J)=X(J)-X(I)
9770 Y(J)=Y(J)-Y(I)
9802 NEXT I
9806 NEXT G
9810 RETURN
9999 END
```

This sequence finishes computing an attack strategy, less marginal and intercept values; then it returns to the control program.

VI. EXAMPLES

In this section we will apply the PATH87 and PATH87A programs to solve the problem of allocating weapons to point targets in a variety of cases selected to illustrate features of the solution method and typical properties of solutions.

A. One-Sided Optimizations; PATH87A

The first four cases are one-sided optimizations involving the allocation of attack weapons to undefended targets. Program PATH87A is preferred for this type of problem, although PATH87 may be used.

The dimensions of each case are given by the number of target types (G1) and the number of attack-weapon types (A1). The cases are arranged in order of increasing complexity.

Case 1: (G1,A1) = (1,1)

The technique can be illustrated by running a simple case with interaction between computer and operator. To begin, we assume that the source program is stored on disk.

```
LOAD PATH87A
READY
```

The operator commands that the source program be loaded into core, and the computer responds when it has done this and is ready for further commands.

```
50 DATA 1,1
60 DATA 100,1
70 DATA .8
RUN
```

The operator modifies the source program and commands that it be run. In this example, the data statements specify 1 target type, 1 attack-weapon type, 100 targets, unit value per target, and 0.8 single shot kill probability.

```
    PATH87A

    G1,A1=  1       1
    T,V=
     100    1
    PKT=
     .8
```

The computer prints a heading and begins the run by printing the data specifications for the record.

```
    P1,P3?
    ? 1,50
```

After initializing, the computer calls for input of P1 and P3 to specify a path segment. By typing "1,50", the operator directs that attack weapon type 1 be changed from its present quantity (initially 0) to 50.

```
          1
             0      50     50
             1      50     10.
    VS= 60.
```

At the end of the path segment, the computer prints a solution, four lines in this case. The first line identifies the target type. The second line says that the elementary strategy $\alpha = 0$ (i.e., no weapons are assigned) is used on each of 50 targets and that the expected surviving value of these is 50. The third line says that the elementary strategy $\alpha = 1$ (i.e., one weapon is assigned to each target) is used on each of 50 targets and that the expected surviving value of these is 10. The last line gives the total expected surviving value.

```
F1,F3?
? -1,0
 1      -.8
```

The computer calls for a new path segment, and the operator types "-1,0", a special input used to direct a printout of marginal values in PATH87A. The computer says that the marginal value for attack weapon type 1 is -.8.

```
F1,F3?
? 1,150
    1
        1      50     16.
        2      50     2.
VS= 18.
F1,F3?
? -1,0
 1      -.16
```

In the same way as before, the number of weapons is increased to 150, with a new solution strategy and a new marginal value of -.16.

```
F1,F3?
? 1,250
    1
        2      50     2.
        3      50     .4
VS= 2.4
F1,F3?
? -1,0
 1      -3.20000E-02
```

The force size is increased to 250, and the marginal value becomes -.032.

```
F1,F3?
? 0,0
CPU-SECS:      0.5
```

A "0,0" input terminates the run, with the computer printing the total central processing time used.

The results are illustrated graphically by the drawdown curve of Figure 13. Here the resource space is the A-axis, divided into regions of length 100 by regional boundaries at the points 0, 100, 200, etc. Elementary solution strategies throughout the first region are $\alpha = 0$ and $\alpha = 1$, throughout the second $\alpha = 1$ and $\alpha = 2$, etc. The solution surface is the drawdown curve itself, a straight line over each region. The tangent line, with its intercept and slope (or marginal value), is uniquely defined over the interior of each region but is indeterminate at regional boundaries, where it may have any slope between those of the two adjacent regions.

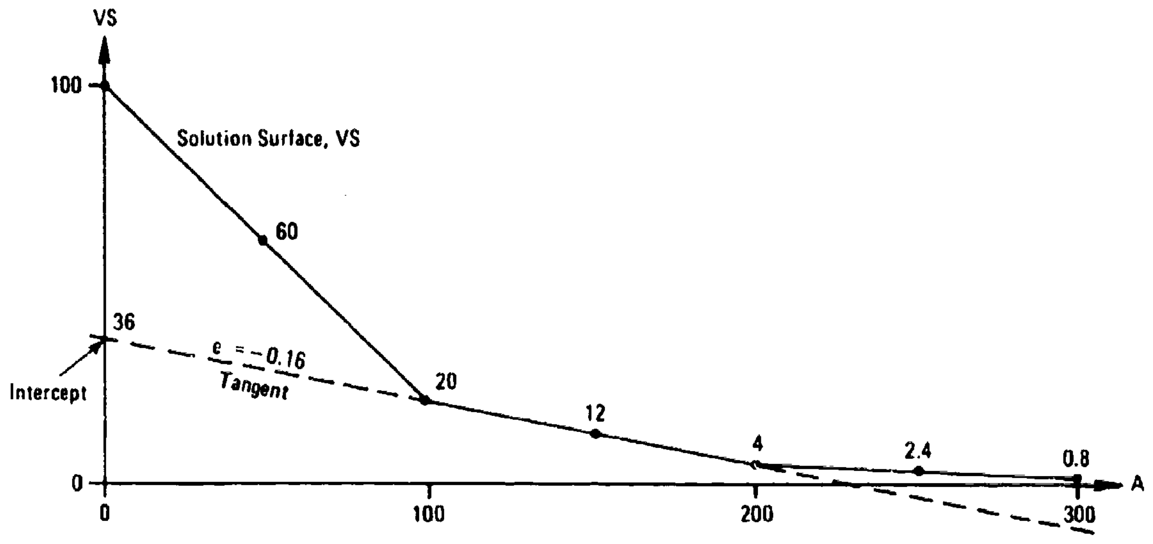


FIGURE 13
SOLUTION OF CASE 1

Case 2: (G1,A1) = (1,2)

A case with two attack weapon types illustrates some additional features. For the following runs, we assume that PATH87A has already been loaded into core.

```
50 DATA 1,2
60 DATA 100,1
70 DATA .8,.6
RUN
```

The operator commands a run with 1 target type, 2 attack weapon types, 100 targets, unit value per target, and 0.8 and 0.6 single shot kill probabilities.

```
PATH87A

G1,A1= 1       2
T,V=
 100    1
PKT=
 .8     .6
```

The computer prints the heading and data record.

```
P1,P3?
? 1,150
    1
        1       50      10.
        2       50      2.
VS= 12.
```

For the first path segment, the input "1,150" produces a solution identical with one of the Case 1 solutions.

```
P1,P3?
? 2,120
    1
        2       0       30      1.2
        1       2       50      1.6
        2       1       20      .32
VS= 3.12
```

The second path segment adds weapon type 2. In the solution, there are now three elementary strategies, given by the vectors (2,0), (1,2), and (2,1).

```
P1,P3?
? -1,0
 1     -4.00000E-02
 2     -2.40000E-02
```

The marginal values for both attack weapons are obtained by the special input "-1,0".

```
P1,P3?
? 0,0
CPU-SECS:    0.5
```

The run ends with a time printout.

Another run with the same data statements can be made simply by commanding RUN.

```
PAIH87A

G1,A1=  1      2
T,V=
 100    1
PKT=
 .3     .6
P1,P3?
? 2,120
     1
        1      80     32.
        2      20.    3.2
VS= 35.2
```

In this example, the order of weapon input is reversed, attack weapon type 2 being raised to 120 on the first path segment.

```
P 1,13?
? 1,150
    1
        2        0        30       1.2
        1        2        50       1.0
        2        1        20       .32
VS= 3.12
P 1,P3?
? 0,0
CPU-SECS:    0.0
```

On the second path segment, attack weapon type 1 is added. In the printout of elementary strategies, the weapon types are arranged in their original numerical order regardless of the path sequence. The solution strategy at the point (150, 120) is identical with that of the preceding run.

In this case, the resource space is the (A_1,A_2)-plane. It is divided into triangular regions as shown by the solid lines on the regional map of Figure 14. The solution surface must be imagined in a third dimension, but the values are shown in parentheses at points of integral density, corresponding to corners of the regions. The solution surface over each region is a plane. The elementary solution strategy at the corner of a region is a uniform strategy, the same on all targets. On a boundary, the solution is a mix of two corner strategies. In a regional interior, the solution is a mix of three corner strategies.

The pattern of regions is associated with the kill probabilities of the problem. In this case, the pattern was determined by making a run along the dashed exploratory path in the figure. The exploratory path is one that goes back and forth, like ploughing a field. We want a printout at every critical point of the path, that is at every regional boundary and at the end of each path segment. For this purpose, the program must be modified.

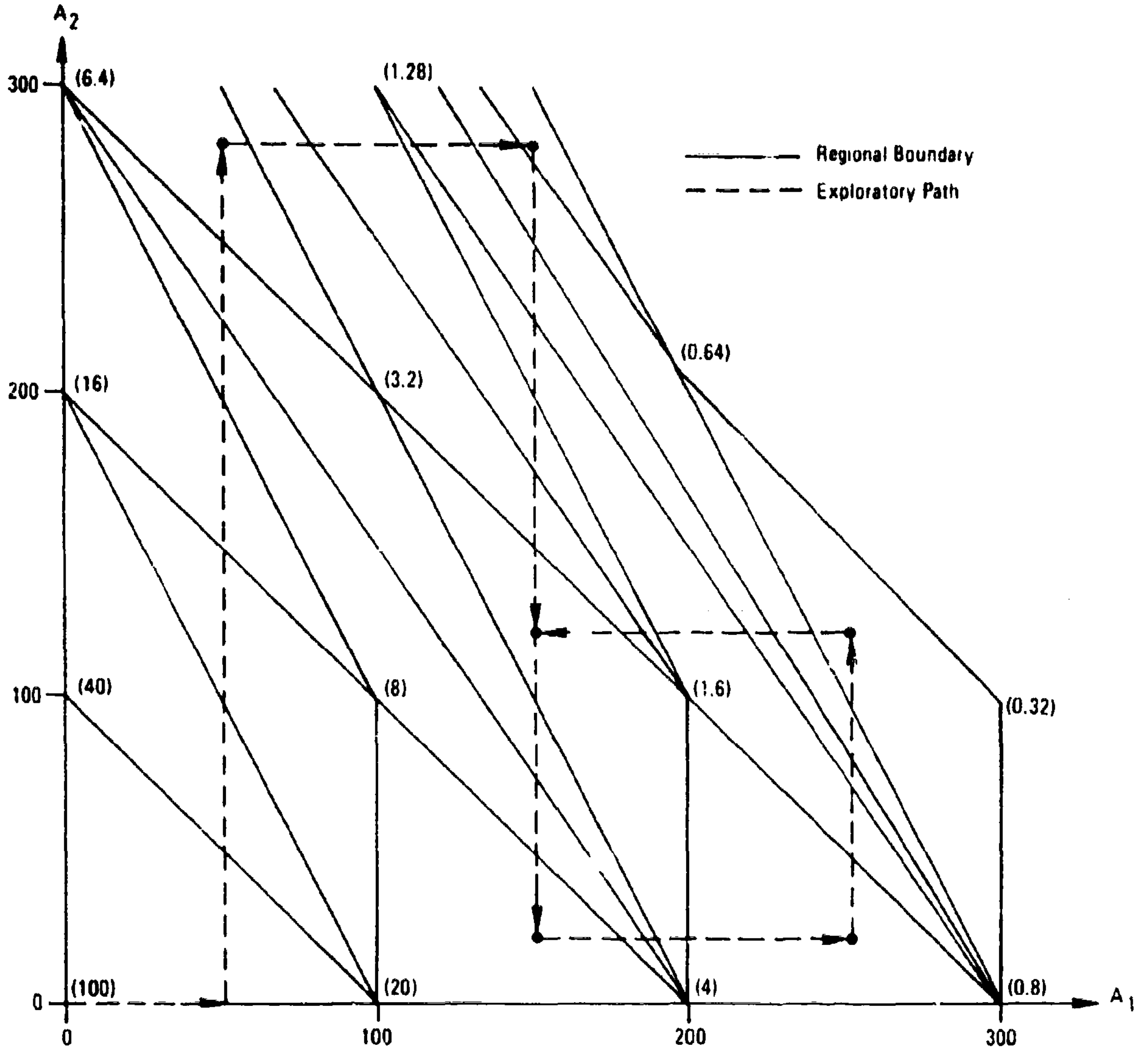


FIGURE 14
REGIONAL MAP OF CASE 2

```
1745 H=H2
1750 GOSUB 5000
1870
RUN
```

The operator adds lines 1745 and 1750 to get a printout at every critical point, and deletes 1870 to delete the usual printout at the end of a path segment.

```
[illegible]

G1,G1= 1       2
G,V=
 100     1
PR1=
 .8      .6
P1,P3?
? 1,50
     1
          0       50      50
          1       50      100.
VS= 60.
```

The solution at (50,0), the end of the first path segment, is identical with that in Case 1.

```
P1,P3?
? 2,280
     1
          0       0       0
          1       0       50      10.
          0       1       50      20.
VS= 30.
```

On the second path segment, running from (50,0) to (50,280), the computer prints a solution strategy at the first regional boundary encountered. The three elementary strategies locate the corner points of a region. Since one of the strategies, i.e., (0,0), applies to zero targets, the path is crossing the opposite regional boundary, specifically at the point (50,50), where VS = 30.

```
      1
         1      0      [illegible]      10.
         0      1      0
         0      2      [illegible]      [illegible].
   VS= 1[illegible].
```

The (0,0) elementary strategy has been deleted and (0,2) is added. The path has reached the point (50,100), where it is again on a regional boundary, and the (0,1) elementary strategy is about to be deleted.

```
      1
         1             [illegible]
         0      2      [illegible]      .
         1      1      [illegible]      [illegible].
   VS= 1[illegible].
      1
         0      2      [illegible]
         1      1      [illegible]      [illegible].
         0      [illegible]      [illegible]      [illegible].
   VS= 7.[illegible]
      1
         1      [illegible]
         0      [illegible]      [illegible]      [illegible].
         2      [illegible]      [illegible]      [illegible].
   VS= 5.79999
```

The path continues, encountering regional boundaries at (50,150), (50,200), and (50,225). Here a small roundoff error appears in the printout of VS, which should be read as 5.8

```
      1
         0      3      47.[illegible]      [illegible].1[illegible]
         2      0      1.5[illegible]E-0[illegible]      [illegible].1[illegible]35[illegible]E-0[illegible]
         1      2      47.[illegible]      1.59[illegible]
   VS= 4.79997
```

At the boundary (50,250), a number of small roundoff errors appear. The correct reading is obviously:

```
            0          3          50          3.2
            2          0           0          0
            1          2          50          1.6
        VS= 4.8.
```

The program continues along the dashed line.

```
       1
          0         3         [illegible]       1.2[illegible]
          1         2         50        1.6
          0         4         [illegible]       .[illegible]
    VS= 3.[illegible]48
```

The path segment ends at an interior point of a region. No roundoff errors appear in the printout.

```
    F 1.1[illegible]
    [illegible] 1.150
       1
          [illegible]         3         0
          1         2         [illegible]       1.[illegible]
          2         4         [illegible]       1.[illegible]
    VS= 2.944
       1
          1         2         1
          0         4         [illegible]       1.[illegible]
          2         1         [illegible]       .[illegible]
    VS= 2.1[illegible]
       1
          0         4         [illegible]-05       3.[illegible]E-07
          2         1         9.[illegible]          .15[illegible]
          1         2         [illegible].     1.1[illegible]
    VS= 1.31[illegible]
       1
          [illegible]         1         1.5[illegible]E-0[illegible]       [illegible].44141E-[illegible]
          1         3         93.[illegible]          1.1[illegible]
          3         [illegible]         [illegible]          [illegible]E-0[illegible]
    VS= 1.24[illegible]
```

The third path segment traverses four regions before reaching a boundary at $(113\frac{1}{3},280)$. Neither the force level nor the strategy is physically feasible, although they are mathematically correct in the path method of solution.

```
     1
        1     [illegible]     1.44141E-04     3.12500E-07
        3     0     43.9999         .351999
        0     5     55.9999         .577433
VS= .96544
     1
        3     0     1.44141E-04     1.95312E-07
        0     5     56.0000         .273009
        2     2     73.333          .409331
VS= .142412
     1
        0     5     20.     .224799
        2     2     64.9999         .440999
        1     4     10      5.11999E-02
VS= .703219
```

The path segment continues to its end at the interior point (150,280). Here the strategy is physically feasible when roundoff errors in the printout are overlooked.

```
1 1,132
2 1,120
     1
        3     5     25.0000         .256
        2     2     74.9999         .479999
        1     4     1.20735E-04     9.77561E-09
VS= .736
     1
        0     5     50      .512
        2     2     1.52588E-05     9.77562E-08
        3     0     50.     .4
VS= .911999
     1
        0     5     1.52588E-05     1.56250E-07
        3     6     25.     .2
        1     3     75.     .959999
VS= 1.16
     1
        3     6     1.52588E-05     1.22070E-07
        1     3     49.9998         .639997
        2     1     50.0002         .800002
VS= 1.44
```

```
      1
         1      3      0
         2      1      75     1.2
         0      4      25     .64
VS= 1.84
      1
         2      1      50.     .799999
         0      4      1.52588E-05     3.90625E-07
         1      2      49.9999      1.6
VS= 2.4
      1
         2      1      20     .32
         1      2      50     1.6
         2      0      30     1.2
VS= 3.12
```

On the fourth path segment, the force level is decreasing, so the value surviving is increasing. At (150,120), the end of the segment, the solution is identical, except for arrangement, with those of the earlier runs of this case.

```
F1,F3?
? 2,20
      1
         2      1      0
         1      2      50     1.6
         2      0      50     2.
VS= 3.6
      1
         1      2      0
         2      0      75     3.
         0      3      25     1.6
VS= 4.6
      1
         2      0      50.0001      2.
         0      3      1.52588E-05     9.76562E-07
         1      1      49.9999      3.99999
VS= 6.
      1
         2      0      50     2.
         1      1      20     1.6
         1      0      30     6.
VS= 9.6
```

The fifth path segment continues in the direction of the fourth, with VS still increasing.

```
F1,F3?
? 1,250
     1
          2        0        40       3.2
          1        1        20       1.6
          1        0        0
VS= 4.8
     1
          2        0        93.3333       3.73333
          1        1        4.57764E-05      3.66211E-06
          0        3        6.66666       .426666
VS= 4.16
     1
          2        0        90.      3.6
          0        3        3.05176E-05      1.95312E-06
          1        2        9.99996       .319999
VS= 3.92
     1
          2        0        40       3.2
          1        2        0
          2        1        20       .32
VS= 3.52
     1
          2        1        30       1.2
          2        1        20       .32
          3        0        50       .4
VS= 1.92
```

The force increases again on the sixth path segment.

```
F1,F3?
? 2,120
     1
          2        0        0
          2        1        50       .8
          3        0        50       .4
VS= 1.2
     1
          2        1        0
          3        0        75       .6
          1        3        25       .32
VS= .919999
```

```
      1
         3       0       83.3334        .666667
         1       3       1.52588E-05       1.95312E-07
         0       5       16.6666        .170666
VS= .837333
      1
         3       0       49.9999        .399999
         0       5       1.52588E-05       1.56250E-07
         2       2       50.0001        .320001
VS= .719999
      1
         3       0       30     .24
         2       2       50     .32
         3       1       20     6.40000E-02
VS= .624
```

The force continues to increase on the seventh path segment, and VS reaches its lowest level of the entire run at the end point (250,120).

```
F1,F3?
? 1,150
      1
         3       0       40     .32
         2       2       60     .384
         3       1       0
VS= .704
      1
         3       0       76     .608
         2       2       6.10352E-05       3.90625E-07
         0       5       23.9999        .245759
VS= .853759
      1
         3       0       59.9999        .479999
         0       5       1.52588E-05       1.56250E-07
         1       3       40.0001        .512001
VS= .992
      1
         3       0       1.52588E-05       1.22070E-07
         1       3       9.99995        .127999
         2       1       90.     1.44
VS= 1.568
```

```
       1
         1       3       1.52588E-05     1.95312E-07
         2       1       93.3333     1.49333
         0       4       6.66667     .176667
VS= 1.664
       1
         2       1       80.0001     1.28
         0       4       1.52588E-05     3.90625E-07
         1       2       19.9999     .039997
VS= 1.92
       1
         2       1       20     .32
         1       2       50     1.6
         2       0       30     1.2
VS= 3.12
```

The last path segment reaches point (150,120) from the ·ight in the map of Figure 14, with the same solution as from each of the other directions.

```
F1,F3?
? 0,0
CPU-SECS:     1.6
```

The exploratory run is terminated with sufficient data to construct the map of Figure 14. By inspecting the figure, we can see that the path has traversed each region at least once and some regions as many as four times.

Although the path method theoretically finds the same solution regardless of the direction of approach to a point, the program PATH87A does not always do so, because of arbitrary limitations of the "floating lid" test-control process discussed in Section IV. The discrepancy is illustrated by the two runs that follow.

```
PATH87A

G1,A1=  1       2
T,V=
 100    1
PKT=
 .8     .6
P1,P3?
? 2,60
    1
        0       40      40
        1       60      24.
VS= 64.
P1,P3?
? 1,150
    1
        1       1       30      2.4
        2       0       60      2.4
        0       3       10      .64
VS= 5.44
P1,P3?
? 0,0
CPU-SECS:    0.6
```

In the first run, the correct VS = 5.44 at (150,60) is obtained by inputting (2,60) and then (1,150).

```
PATH87A

G1,A1=  1       2
T,V=
 100    1
PKT=
 .8     .6
P1,P3?
? 1,150
    1
        1       50      10.
        2       50      2.
VS= 12.
P1,P3?
? 2,60
    1
        2       0       50      2.
        1       1       40      3.2
        1       2       10      .32
VS= 5.52
```

In the second run, with the input order reversed, the answer VS = 5.52 is not optimal, because the elementary strategy (0,3) was barred from testing when the boundary at (150,50) was crossed, since the "floating lid" would not permit testing either weapon at a level higher than 1 above its previous actual maximum.

```
F1,F3?
? 2,20
    1
        2       0       50      2.
        1       1       20      1.6
        1       0       30      6.
VS= 9.6
F1,F3?
? 2,60
    1
        2       C       60      2.4
        1       1       30.     2.4
        C       3       9.99997         .639998
VS= 5.44
F1,F3?
? 0,0
CPU-SECS:     0.6
```

When the same run is continued by withdrawing, input (2,20), and then readvancing, input (2,60), the correct solution is obtained, because the lid has floated high enough to permit the elementary strategy (0,3) to be tested the second time the boundary is crossed. The "floating lid" process needs improvement in order to achieve absolute precision, as well as to save time.

<u>Case 3· (G1,A1) = (1,5)</u>

Some additional features can be illustrated by a case with five attack-weapon types.

```
50 DATA 1,5
60 DATA 100,1
70 DATA .8,.6,.5,.3,.1
RUN

PATH87A

G1,A1= 1      5
T,V=
 100    1
PK(=
 .8     .6     .5     .3     .1
P1,P3?
? 1,79
     1
         0      21     21
         1      79     15.8
VS= 36.8
P1,P3?
? 2,79
     1
         1      0      42     8.4
         0      2      21     3.36
         1      1      37     2.96
VS= 14.72
P1,P3?
? 3,79
     1
         1      1      0      58     4.64
         1      0      1      21     2.1
         0      1      2      4.99997       .499997
         0      1      3      16.    .800001
VS= 8.03999
```

The resource space is now three dimensional and is divided into tetrahedral regions with four corners and triangular faces. The solution contains four elementary strategies.

```
P1,P3?
? 4,79
     1
         1      0      1      1      15.    1.05
         0      3      0      0      5.00002       .320001
         1      1      0      1      64     3.584
         0      0      4      0      16.    .999997
         0      0      3      2      1.52583E-05    9.34601E-07
VS= 5.954
```

Each new weapon type adds a dimension to the resource space and an elementary strategy to the solution. The general rule for one-sided optimizations is that the number of elementary strategies is equal to the number of object types plus the number of resource types.

```
F1,F3?
? 5,79
    1
        1      1      0      1      0      57.9999     3.24799
        0      0      4      0      1      12.0003     .675015
        0      3      0      0      2      6.99992     .362876
        1      0      1      1      3      17     .867509
        0      0      3      2      1      1.99995     .110247
        1      0      2      0      0      3.99997     .199993
VS= 5.46363
```

The final resource space has five dimensions and is divided into six-cornered regions called simplexes. Roundoff errors in the printout of the target column can be corrected to 58, 12, 7, 17, 2, and 4, giving an integral solution strategy.

```
F1,F3?
? -1,0
 1     -8.57613E-02
 2     -4.88513E-02
 3     -3.69425E-02
 4     -1.90338E-02
 5     -5.02627E-03
F1,F3?
? 0,0
CPU-SECS:    7.4
```

The marginal values, or slopes of the tangent hyperplane, are all very small at this attack level.

We cannot guarantee that the final solution strategy is absolutely the best integral strategy, since the "floating lid" test process may have arbitrarily excluded a better elementary strategy than one of those selected. However, we can find a lower bound for VS by supposing that the aggregate kill potential of the force could be spread uniformly over all the targets. This lower bound is VS = 5.4589, computed by hand. Clearly the integral strategy of the computer run couldn't be improved very much, if at all.

<u>Case 4: (G1,A1) = (5,3)</u>

Multidimensionality may extend to both target and weapon types, as in the following run.

```
50 DATA 5,3
60 DATA 1,10,15,8,10,6,20,5,50,2
70 DATA .8,.5,.5,.7,.6,.3,.7,.6,.4,.9,.5,.3,.4,.2,.5
RUN

FATHR7A

G1,A1= 5       3
T,V=
 1      10
 15     8
 10     6
 20     5
 50     2
PKT:
 .8     .5     .5
 .7     .6     .3
 .7     .6     .4
 .9     .5     .3
 .4     .2     .5
```

The five target groups consist of 1 target of value 10, 15 targets of value 8, etc., a total of 96 targets of aggregate value 390. The 3 weapon types have different kill probabilities against targets of the different groups. The first weapon type is generally the best, but the third weapon type is best against group 5 targets.

```
F1,F3?
? 3,50
    1
        2        1        2.5
    2
        2        15       58.8
    3
        1        10       36.
    4
        0        12       66
        1        8        28.
    5
        0        50       100.
VS= 285.3
```

On the first path segment, 50 type 3 weapons are used. There are two elementary strategies on target group 4, but only one elementary strategy on each of the other groups. In the sense of Figure 13, that means there is an interior strategy on group 4 and corner strategies on the other groups.

```
F1,F3?
? -1,0
 3      -1.5
```

The marginal value is determined by the interior strategy on group 4.

```
F1,F3?
? 2,50
    1
        0        3        1        1.25
    2
        1        0        0
        2        0        15       19.2
    3
        0        2        10       21.0
    4
        1        0        20       50
    5
        0        0        23       46
        0        1        27       27
VS= 165.05
```

Reproduced from best available copy.

On the second path segment, 50 type 2 weapons are added to the force, and some reallocation of type 3 weapons takes place. In the sense of Figure 14, there are corner strategies on groups 1, 3, and 4, edge strategies on groups 2 and 5, and no interior strategies.

```
F1,F3?
? -1,0
 2     -1.42
 3     -1
```

The marginal value for weapon type 2 is determined by the strategy on target group 2, and that for weapon type 3 by the strategy on target group 5.

```
F1,F2?
? 1,50
    1
       2       0       0       1        .4
    2
       1       2       0       15       5.76
    3
       2       0       0       3.33333      1.8
       0       3       0       6.66667      2.56
    4
       1       0       0       13.6667      6.83333
       2       0       0       6.33334      .316667
    5
       0       0       1       50       50
       0       0       2       0
VL= 67.67
```

Now 50 type 1 weapons are added, and the other weapons are reallocated. We note that all the type 3 weapons are used against target group 5, where they are more effective than types 1 and 2. There are corner strategies on two groups and edge strategies on three. Boundary strategies of some sort are typical in multigroup cases.

The strategies on groups 3 and 4 are not physically feasible, since they call for the use of an elementary strategy on a fractional number of targets. If desired, the solution can be adjusted by hand to a feasible one, for instance:

```
                 3
                      2         0         0         3        1.62
                      0         3         0         6        2.304
                      1         2         0         1         .288
                 4
                      1         0         0        14        7.
                      2         0         0         6         .3
          VS= 67.672,
```

which is close to the printed value. Ordinarily, the discrepancy can be ignored, since it is a local one, as is shown by the following path segment.

```
P1,P2?
? (2,51)
     1
         2       0       0       1       .4
     2
         1       2       0       15      5.76
     3
         2       0       0       3       1.62
         0       3       0       7.      2.633
     4
         1       0       0       13.     6.5
         2       0       0       7.      .35
     5
         0       0       1       50      50
         0       0       2       0
VS= 67.315
```

If one type 2 weapon is added to the force, the strategy becomes physically feasible. The changes taking place show that physically feasible strategies will result from 48, 51, 54, etc., type 2 weapons and infeasible ones from 49, 50, 52, 53, etc. The user will have to decide whether extra precision is worth the trouble.

```
F1,F3?
? -1,0
 1      -.45
 2      -.352
 3      -.5
F1,F3?
? 0,0
[illegible]      2.2
```

The marginal values show that, at the final force level (50,51,50), one extra type 3 weapon would add more to force effectiveness than one of either of the other types.

B. Two-Sided Games; PATH87

The next four cases are two-sided games involving the allocation of defense weapons as well as of attack weapons. Program PATH87 must be used for these.

In addition to the dimensions G1 and A1, we must also specify the number of defended-target types (G4) and the number of defense-weapon types (D1).

Case 5: (G1,G4,A1,D1) = (1,1,1,1); Perfect Weapons

This perfect-weapon case is analyzed in Reference 1, where explicit solution formulas are given. It is included here to illustrate the numerical solution by the PATH87 program.

```
50 DATA 1,1,1,1
60 DATA 100,1
70 DATA 1
80 DATA 1
RUN
```

The first line of data input specifies the four dimensions. The added fourth line specifies the probability of intercept by the defense weapon type. The other lines are the same for both PATH87 and PATH87A.

```
PATH87

G1,G4,A1,D1= 1       1       1       1
T,V=
 100    1
PKT=
 1
PI=
 1
P1,P3?
? 1,50
```

After printing a heading and recording data, the computer calls for the operator to input a path segment. Some attack must be specified before any defense, so the operator inputs 50 type 1 attack weapons.

```
     1       50
         0       50.
         1       50.
VS= 50.
```

The form of the strategy printout differs in two respects from that of PATH87A. Here the first line shows an intercept value of 50, and the elementary strategy lines omit the expected-survival column of PATH87A.

```
P1,P3?
? -1,175
```

The operator inputs 175 type 1 defense weapons (the - sign indicates defense).

```
1      -.225
    1       100
        0       22.5
        1       22.5
        2       22.5
        3       22.5
        4       10.
VS= 43.75
```

The computer prints the defense solution strategy at the end of the second path segment; that is, at the point (50;175). The first line says attack weapon type 1 has a marginal value of -.225—that is, the slope of the tangent plane in the direction of the attack coordinate. The second line says that on target group 1 the intercept value is 100—that is, the ordinate [i.e., value at the point (0;175)] of the plane tangent to the solution surface at (50;175). The next five lines give elementary defense strategies and the number of targets for each. The last line gives the expected value surviving at the point (50;175) as 88.75, which is computed as the intercept value plus the marginal value multiplied by the number of attack weapons.

The weapon allocation itself exhibits the equal steps, i.e., 22.5 targets, characteristic of perfect weapon cases. When these fractional allocations are viewed as mixed strategies, they are physically feasible, since numerous convex combinations of integral allocations can be found that are precisely equivalent. Perhaps the simplest combination consists of the strategy

0	22
1	23
2	23
3	22
4	10

with a frequency of .5 and the strategy

0	23
1	22
2	22
3	23
4	1C

with a frequency of .5.

```
F1,F3?
? 1,50
```

The operator calls for an attack strategy printout by inputting an attack weapon at no change from its current force level.

```
 1      5.00000E-02
    1      80
       0      80
       1      5.
       2      5.
       3      5.
       4      5.
VS= 88.75
```

Although the attack force level is unchanged, the attack solution strategy has changed. Before, there were 50 targets attacked. Now that a defense has been introduced, the attack is concentrated in an equal step strategy on only 20 targets. Notice that the marginal value per defense weapon is .05, the intercept value at (50;0) is 80, and VS = 80 + 175 x .05 = 88.75, as it should.

A complete solution at the point (50;175) is provided by this attack strategy and the preceding defense strategy. The point happens to lie in a region that we call defense dominant, where some targets are not attacked.

```
F1,F3?
? 1,275
 1      .224999
    1      0
       4      22.5000
       1      22.4999
       2      22.4999
       3      22.4999
       5      9.99964
VS= 39.3749
```

```
F1,F3?
? -1,175
 1     -.175005
    1      87.5013
        0      29.9993
        1      17.5005
        2      17.5005
        3      17.5005
        4      17.4992
VS= 39.3748
```

A complete solution at the point (275;175) is obtained by first increasing the attack force to 275 and then calling for a defense printout. This point lies in a region of attack dominance, since every target is attacked by at least one weapon. The roundoff arrors are annoying, but they can be easily adjusted in this case, for instance by reading both 22.5006 and 22.4999 as 22.5.

```
F1,F3?
? 0,0
CPU-SECS:     1.7
```

The run is terminated in the usual fashion.

The same problem can be run with slightly more generality as a density problem by normalizing inputs as in the following run.

```
20 DATA 1,1
RUN
```

The data change specifies 1 target instead of 100.

```
PATH87

G1,G4,A1,D1=  1        1        1        1
T,V=
 1       1
PKT=
 1
P1=
 1
P1,P3?
?  1,.5
```

The first path input is a density of attack of .5 instead of 50 weapons. Subsequent inputs will be scaled the same way.

```
       1       .5
           0       .5
           1       .5
VS= .5
P1,P3?
?  -1,1.75
 1      -.225
       1       1
           C       .225
           1       .225
           2       .225
           3       .225
           4       .1
VS= .8875
P1,P3?
?  1,.5
 1         5.00000E-02
       1       .8
           C       .8
           1       5.00000E-02
           2       5.00000E-02
           3       5.00000E-02
           4       5.00000E-02
VS= .8875
```

```
F1,F3?
? 1,2.75
 1      .225
    1      0
       4       .225
       1       .225
       2       .225
       3       .225
       5       .1
V5= .39375
F1,F3?
? -1,1.75
 1      -.175
    1      .874999
       0       .3
       1       .175
       2       .175
       3       .175
       4       .175
V5= .39375
F1,F3?
? 0,0
CPU-SECS:     1.6
```

The results of the density run should be compared item by item with those of the preceding run. The solution strategies and values can be scaled to apply to any number of targets of any unit value. An extra benefit of the density technique is the reduction of roundoff errors.

Case 6: (G1,G4,A1,D1) = (1,1,1,1); Imperfect Weapons

This is the imperfect weapon case which is analyzed in Reference 2. That report presents solutions in closed form, but the formulas involve tedious summations that are best handled by computers.

The use of PATH87 to find a point solution can be illustrated by a density run with (PKT,PI) = (0.8,0.75).

```
50 DATA 1,1,1,1
60 DATA 1,1
70 DATA .8
80 DATA .75
RUN

PATHR7

G1,G4,A1,D1= 1       1       1       1
T,V=
 1      1
PKT=
 .8
PI=
 .75
P1,P3?
? 1,5.666666
     1      1.49334E-04
         5      .333334
         6      .666666
VS= 1.49334E-04
P1,P3?
? -1,3.666666
 1      -5.69525E-02
     1      .469857
         0      .410667
         6      .173805
         5      9.26959E-02
         7      .289675
         4      3.31568E-02
VS= .147127
P1,P3?
? 1,5.666666
 1      3.99547E-02
     1      6.26740E-04
         5      .121933
         6      .152414
         7      .225917
         4      .359197
         8      .140538
VS= .147127
P1,P3?
? 0,0
CPU-SECS:    1.7
```

The solution strategies at the point (17/3;11/3) are identical with the canonical strategies pictured in Reference 2 (Figure 23, p. 172) except for notational differences.

PATH87 can be used for exploratory runs, as in the following example, which also illustrates certain noteworthy features of the solution process.

```
1545 GOSUB 5300
1745 GOSUB 5300
1870
5300 IF H2<=H3 THEN 5330
5310 H=H3
5320 GOTO 5340
5330 H=H2
5340 GOSUB 5600
5350 RETURN
```

The program must be modified to print a complete spectrum of strategies along the path.

```
50 DATA 1,1,1,1
60 DATA 1,1
70 DATA .8
80 DATA .75
RUN
PATH87

G1,G4,A1,D1= 1        1        1        1
T,V=
 1        1
PKT=
 .8
PI=
 .75
P1,P3?
? 1,.1
```

The first path segment runs from the origin to the point (.1;0).

```
    1     -.8
       1       1
           1
  VS= 1
```

The first strategy is a defense strategy at the origin, in the usual form except for the third line, which is an intentionally incomplete representation of the null allocation at the origin. The only important line is the one that specifies the initial marginal value for the attack as -.8.

```
       1      .92
          2      .9
          1      .1
  VS= .92
```

Next is an attack strategy, the rule being an alternation of attack and defense strategies in the spectrum along a path segment. As no defense weapon has yet been introduced, the printout of marginal value is suppressed and the first print line shows an intercept value of .92, the same as VS. Since the strategy allocates all prescribed weapons, the path segment is terminated.

```
  F1,F32
  2 -1,1
   1       5.99999E-02
       1      .92
          2      .9
          1      .1
  VS= .92
```

The input for the next path segment calls for raising the defense force to a density of 1 weapon per target. Whenever a weapon type is introduced, the first strategy is an opposition strategy, so here an attack strategy is printed. It is the same as the strategy terminating the preceding segment except that it now contains a marginal value of .06 for the weapon type just introduced.

```
 1     -.444444
     1      1
         0       .407407
         1       .592593
 VS= .955556
```

Next is a defense strategy in standard form. Since it occurs at a force level of .592593, which is before the end of the path segment, we know that a regional boundary has been encountered at this force level. As both elementary strategies, δ = 0 and δ = 1, are shown to apply to more than zero targets, neither can be deleted; so we infer that a new elementary attack strategy must be added to the Q-basis at the boundary. The value of the game at the point (.1;.592593) is .955556.

```
 1       1.84614E-02
     1      .944615
         0       .938461
         1       2.30769E-02
         2       3.84615E-02
~~ VS= .944615~~
```

The printout of the attack strategy on the boundary shows that the elementary strategy, α = 2, has been added, and the attack is now concentrated on a smaller fraction of the target system. The VS printout here should be disregarded, because the program computes VS using the defense force level recorded in the ($) matrix, and this matrix is not updated until the end of the path segment. Actually, this attack strategy is a solution strategy for a range of defense force levels, beginning at .592593, where VS = .944615 + (.592593)·(.0184614) = .955555, the same as printed for the preceding defense strategy. The range includes the following defense force level, which we can anticipate by computing VS = .944615 + (1.0)(.0184614) = .963076.

```
    1      -.369231
        1        1
            0          .282052
            1          .435897
            2          .282051
  VS= .963077
```

The path segment ends with a defense strategy that includes the elementary strategy $\delta = 2$. A complete solution at the end point (.1;1) consists of this defense strategy, the preceding attack strategy, and the value of the game, which is computed here as VS = 1 + (.1)(-.369231) = .963077.

The course of events along the path segment can be rationalized in the following terms:

First—at the beginning, all of the attack weapons are on separate targets, since there is no defense.

Second—As defense weapons are added, they are allocated to separate targets with an expected saving of .06 targets per weapon.

Third—When the defended fraction of targets reaches .592593, it becomes just as good for the attack to put 2 weapons as 1 on a target.

Fourth—At this point, the attack shifts to a strategy that kills just as many targets but offers the defense the smallest possible saving for any additional weapons, i.e., .0184614. The new strategy is so balanced between the targets attacked with 1 weapon and those attacked with 2 weapons that the defense can do equally well by defending 1 target with 2 weapons as by defending 2 targets with 1 weapon each.

Fifth—The defense strategy does not change at this point, but as new defense weapons are added to the force, the balance between the elementary strategies $\delta = 1$ and $\delta = 2$ is maintained so that the attack is unable to profit by changing allocation.

It should be recognized that each strategy printed out along the path segment is a solution strategy when paired with the preceding strategy of the opposition, the following strategy of the opposition, or any convex combination of the preceding and following strategies. Each printed attack strategy is valid for some range of this path segment, and convex combinations of them represent transitions at a boundary point. Each printed defense strategy is valid only at a point of the path segment, and convex combinations of them represent solutions at intermediate points. In this sense, the printed strategies constitute a complete spectrum of solutions along the path segment.

```
[illegible] 1,1.3[illegible]
[illegible] 1,2.6
  1        .3
     1        .1
          0        4.1[illegible]33E-07
          1        .375
          2        .625
[illegible]  .4
```

The path input calls for the attack density to be increased from .1 to 2.6. Since no new weapon type is introduced, the first strategy printed is an attack strategy at the critical point (1.625;1), the first boundary encountered on the path segment. Here all targets come under attack, and the elementary strategy $\alpha = 0$ leaves the Q-basis. The path is leaving a region of defense dominance and entering a region of attack dominance.

```
 1      -.246154
    1       .799999
       0       .410257
       1       .179487
       2       .410256
```
~~VL= [illegible]~~

At the boundary, the defense makes a transition to a strategy for the next region. The new strategy concentrates on defending fewer targets and offers a numerically-reduced marginal value to the attack. Since no elementary strategy is zeroed out, we infer that the next attack strategy will contain a new elementary strategy.

```
 1        .184616
    1        6.15386E-02
       2       .253465
       1       .23077
       3       .480764
  VL= .246155
```

The next attack strategy includes α = 3, as inferred. The attack force is now at a critical value of 2.25. No elementary strategy is zeroed out, so we infer that the next defense will have an added elementary strategy.

```
 1      -.16
    1       .006153
       0       .589743
       1      -1.19209E-07
       2       .230769
       3       .179486
```
~~VL= .594153~~

The defense does have four elementary strategies, but δ = 1 is immediately zeroed out during the transition at the boundary. The defense is now more concentrated, and the marginal value is reduced in magnitude.

```
   1        .134615
        1         2.46154E-02
              2        .519232
              1        0
              3        .480768
  VS= .269231
```

The path reaches another boundary at the critical attack size of 2.480768, where α = 1 is zeroed out and every target is attacked by at least two weapons.

```
   1       -.11826
        1        .502606
              0        .62319
              2        .136434
              3        .346376
```
[illegible]

The defense transitions to a more concentrated strategy, again reducing the magnitude of the marginal value.

```
   1        .172173
        1         2.29565E-02
              2        .424630
              3        .431519
              4        3.42419E-02
  VS= .19513
```

At (2.6;1), the end point of the path segment, a complete solution is given by this attack strategy and the preceding defense strategy. The VS can be computed from either one,

$$\begin{aligned} VS &= .502606 + (2.6)(-.11826) \\ &= .0229565 + (1)(.172173) \\ &= .19513 \quad . \end{aligned}$$

This is the same value given in Reference 2 (page 159). The spectrum of strategies on the preceding path segment is identical, except for notation, with part of the spectrum pictured in Reference 2 (Figure 18, page 157). For the next path segment, we will now run a portion of the other spectrum appearing in Reference 2 (Figure 19, page 158).

```
F1,F3?
? -1,3.5
 1      -.16
    1       .65
        0       .466667
        2       .2
        3       .333334
VS= .264
```

The path input calls for increasing the defense to a density of 3.5 weapons per target. The first defense strategy is at the critical density of 1.4.

```
 1      .152307
    1       5.07593E-02
        2       .23793
        3       .352482
        4       .218754
        1       .190336
~~VS= .[illegible]~~
 1      -.235294
    1       1.
        0       5.88236E-02
        2       .294117
        3       .490196
        1       .156863
VS= .388235
```

A new attack and a new defense have four elementary strategies each.

```
 1      .111863
    1       .140514
        2       .174694
        3       .215364
        4       .363942
        1       .139753
        0       .103246
~~VS= .256017~~
```

Here the attack transitions to a more concentrated strategy, leaving some targets unattacked. The path is entering a defense dominant region.

```
 1      -.2
    1       .999999
        0       5.96046E-08
        2       .187498
        3       .338539
        1       8.33322E-02
        4       .390631
VS= .480001
```

This critical defense strategy displays a new phenomenon. When the defense force has increased to a size of 3.0365, it becomes worthwhile to defend every target with at least one weapon, so the δ = 0 line leaves the Q-basis.

```
 1      .105076
    1       .161091
        2       .164103
        3       .209124
        4       .341882
        1       .248337
        0     -9.53674E-07
~~VS= .166111~~
```

The attack responds to the new situation by attacking every target at least once, and reducing the marginal value for new defense weapons, which can only be added above the 1 level.

```
 1      -.188235
    1       .988233
        1       5.88263E-02
        2       .156861
        3       .294115
        4       .490197
VS= .498822
```

The defense increases to a point where it pays the attack to use 5 weapons on some targets.

```
    1      5.50417E-02
       1       .321825
           2       8.66043E-02
           3       .107502
           4       .13438
           1       .448149
           5       .223964
  ~~VS= .376644~~
```

The attack transitions to a strategy where more targets are attacked by 1 weapon only, the balance of the force being concentrated on fewer targets.

```
    1     -.178455
       1       .978452
           1       3.84499E-02
           2       .131392
           3       .257185
           4       .437666
           5       .135307
  VS= .51447
  P1,P3?
  ? 0,0
  CPU-SECS:    2.1
```

The path segment ends at (2.6;3.5), an interior point of a region. The run is terminated after 2.1 seconds of central processing time, including 1.3 seconds of time to compile the program before running.

Figure 15 shows a regional map for these weapon types. The general pattern of interlocking rectangular regions is characteristic for one imperfect attack-weapon type and one imperfect defense-weapon type although the precise locations of the boundaries vary with the probabilities of kill and intercept. The pattern is the same as that shown in Figure 17 of Reference 2. The solution surface over each rectangular

The defense increases to a point where it pays the attack to use 5 weapons on some targets.

```
      1      5.50417E-02
          1      .321825
             2      8.66043E-02
             3      .107502
             4      .13438
             1      .448145
             5      .223964
    ~~VS= .276867~~
```

The attack transitions to a strategy where more targets are attacked by 1 weapon only, the balance of the force being concentrated on fewer targets.

```
      1    -.178455
          1      .978452
             1      3.84499E-02
             2      .131392
             3      .257185
             4      .437660
             5      .135307
    VS= .51447
    P1,P3?
    ? C,C
    CPU-SECS:    2.1
```

The path segment ends at (2.6;3.5), an interior point of a region. The run is terminated after 2.1 seconds of central processing time, including 1.3 seconds of time to compile the program before running.

Figure 15 shows a regional map for these weapon types. The general pattern of interlocking rectangular regions is characteristic for one imperfect attack-weapon type and one imperfect defense-weapon type although the precise locations of the boundaries vary with the probabilities of kill and intercept. The pattern is the same as that shown in Figure 17 of Reference 2. The solution surface over each rectangular

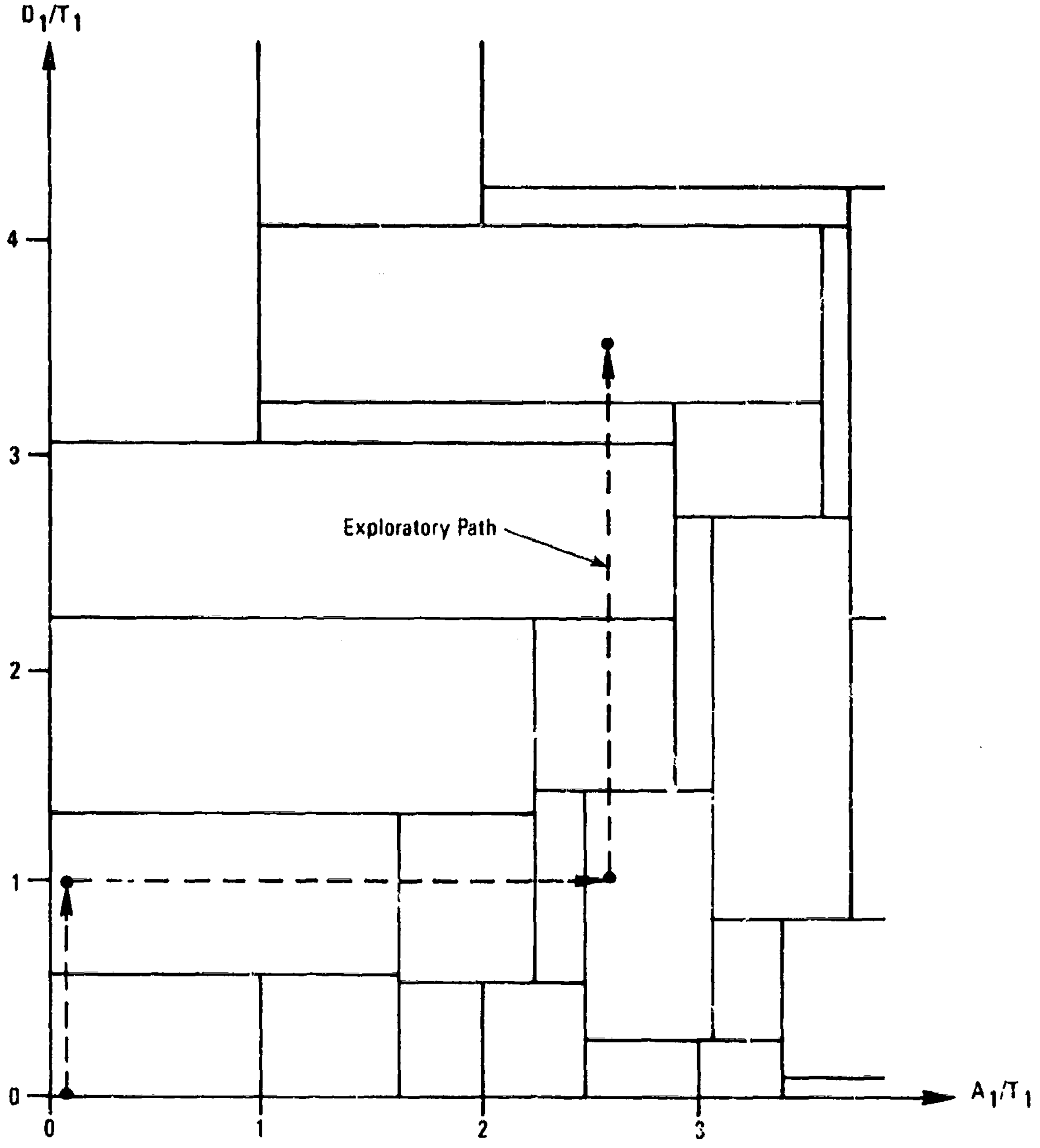


FIGURE 15
REGIONAL MAP OF CASE 6

region is a hyperbolic paraboloid with rulings in the coordinate directions, as discussed in Reference 2. These features contrast with the triangular regions and planar solution surfaces already shown for cases with two attack-weapon types.

Case 7: (G1,G4,A1,D1) = (1,1,2,2)

The following density run illustrates the multiweapon case.

```
50 DATA 1,1,2,2
60 DATA 1,1
70 DATA .6,.8
80 DATA .9,.75
RUN
```

The attack weapons are conventionally arranged in order of increasing PKT, 0.6 and 0.8; the defense weapons in order of decreasing PI, 0.9 and 0.75.

```
PATH87

G1,G4,A1,D1= 1       1       2       2
T,V=
 1       1
PKT=
 .6      .8
PI=
 .9      .75
P1,P3?
? 2,.1
     1       .92
         0       .9
         1       .1
VS= .92
```

```
F1,F3?
? -2,1
 2      -.369231
     1      1
          0        .282052
          1        .435895
          2        .282053
VS= .963077
```

The first two path segments input attack weapon type 2 and defense weapon type 2. As these are the same types used in Case 6, the solution at the point (0,.1;0,1) is also the same as in the exploratory run of that case.

```
F1,F3?
? -1,1
 2      -.224471
     1      1
          0      0        .142968
          0      1        .203141
          0      2        .14297
          1      1        .100602
          2      1        .331555
          3      1        7.87643E-02
VS= .977553
```

At the point (0,.1;1,1) the defense solution strategy is stepped in form, with some targets being undefended and some defended by from 1 to 4 weapons. The better interceptor is concentrated on about half the targets and the poorer interceptor spread out over about 86 percent of the targets.

```
F1,F3?
? 2,.6
 1        4.14715E-02
 2        3.10160E-02
     1        .79243
          0        .782091
          1        3.37700E-02
          2        5.09300E-02
          3        5.34532E-02
          4        7.47475E-02
VS= .865317
```

```
F1,F3?
? 1,.6
 1        8.61790E-02
 2        6.43138E-02
     1        .585421
         0          0          .561123
         0          1          7.08992E-02
         0          2          .103544
         1          2          5.75764E-02
         3          1          .142717
         2          1          5.71372E-02
VS= .735914
```

At the point (.6,.6;1,1), the attack strategy shows the poorer weapon concentrated on about one fourth of the targets and the better one spread out over about 44 percent of the targets. The levels of attack range from 0 to 4 weapons per target.

```
F1,F3?
? -1,1
 1      -.211658
 2      -.229084
     1          1
         0          0          .164347
         0          1          .2117
         0          2          4.45803E-02
         1          1          .15474
         2          1          .300362
         2          2          .119771
VS= .735915
```

A printout of the defense strategy shows only minor changes resulting from the increased attack.

```
F1,F3?
? 0,0
CPU-SECS:     3.2
```

The run is terminated.

The resource space is four dimensional, so we can't show a regional map of it. Actually, we don't know enough to discuss the regions in much detail. The space is bounded by six coordinate planes: a map of the (A_1,A_2)-plane has triangular regions like Figure 14 with the axes reversed; a map of the (A_2,D_2)-plane is the same as Figure 15; maps of the (A_1,D_1)-, (A_1,D_2)-, and (A_2,D_1)-planes have rectangular regions similar to those of Figure 15; and a map of the (D_1,D_2)-plane cannot be defined since VS = 1, regardless of the defense strategy, when there is no attack. Intuitively, we expect the interior regions to have both rectangular and triangular faces, and this is confirmed by a few exploratory paths, which have also shown that some of the multidimensional regions overlap.

Case 8: (G1,G4,A1,D1) = (10,4,1,1)

The multiple-target-type case is illustrated by the following run.

```
50 DATA 10,4,1,1
60 DATA 10,10,10,9,10,8,10,7,10,6,10,5,10,4,10,3,10,2,10,1
70 DATA .55,.6,.65,.7,.75,.8,.85,.9,.95,1
80 DATA .75
RUN
PATH87       15:33

G1,G4,A1,D1= 10      4       1       1
T,V=
 10     10
 10     9
 10     8
 10     7
 10     6
 10     5
 10     4
 10     3
 10     2
 10     1
```

```
PKT=
 .55
 .6
 .65
 .7
 .75
 .8
 .85
 .9
 .95
 1
PI=
 .75
```

There are 10 target groups, only four of which are defended (by convention, the first four). Each group has 10 targets, the total number of targets is 100, and the aggregate value is 550. The single attack weapon type has a different probability of kill for each group of targets.

```
P1,P3?
? 1,165
      1       9.1125
          3       10
      2       10.0801
          2       5
          3       4.9999
      3       9.8
          2       10
      4       6.3
          2       10
      5       3.75
          2       10
      6       10
          1       10
      7       6.
          1       10
      8       3.
          1       10
      9       1.
          1       10
      10      0.
          1       10
VS= 59.0426
```

On the first path segment, the attack lays down 165 weapons as if all the targets were undefended. The marginal value to the attack has not been printed, but it can easily be computed from the strategy on target group 2 as $(9)(1-.6)^2$ $(-.6) = -.864$.

```
P1,P32
2 -1,55
 1     -1.25112
    1       72.7936
       0        4.91072
       3        1.88492
       2        .598212
       4        2.59995
    2       58.3854
       0        6.25423
       2        1.13042
       3        2.56535
    3       56.7837
       0        5.25346
       2        1.70861
       3        2.97292
    4       55.4063
       0        4.04214
       2        2.27586
       3        3.50121
       1        .180771
    5       27.5112
       0        10
    6       22.5112
       0        10
    7       18.5112
       0        10
    8       15.5112
       0        10
    9       13.5112
       0        10
    10      10
       0        10
VS= 144.487
```

On the second path segment, the defense allocates 55 weapons. VS increases from 59.0426 to 144.487. The marginal value to the attack increases to -1.25112, as determined by the defense strategy on the first four groups.

```
P1,P3?
? 1,165
 1      1.27356
    1      9.52689
       3      1.98503
       4      4.51227
       2      2.83698
       5      .665722
    2      7.24513
       2      3.29035
       3      2.78107
       4      3.92859
    3      5.38989
       2      3.8704
       3      3.86205
       4      2.26757
    4      7.21369
       2      2.07259
       3      5.07863
       1      2.20604
       4      4.27437E-02
    5      15
       1      10
    6      10
       1      10
    7      6.
       1      10
    8      3.
       1      10
    9      1.
       1      10
    10     10
       0      10
VS= 144.436
```

A printout of the attack strategy at the point (165;55) shows target group 10 no longer attacked and target group 5 under reduced attack. The shift of 20 weapons from these two groups to the defended groups takes advantage of the higher marginal values provided by the defense. Some shift of the attack from undefended to defended targets is usual as the defense begins to build up. If the defense gets very strong, however, the marginal values may decrease, and some or all of the attack weapons may shift back to the undefended targets. The operator of the program should be alert for a shift of all the attack weapons to undefended targets, since the resulting indeterminate defense could lead to a failure of the program. The program ought to have a procedure for terminating the path segment in such an event, but the current version does not.

```
P1,P3?
? 0,0
CPU-SECS:    4.2
```

The run terminates.

VII. OTHER APPLICATIONS

The illustrative cases in Section VI involved no substantial changes to the PATH87 and PATH87A programs, since only data input and, in a few examples, printout were affected.

Other types of resource allocation problems can be run if the programs are modified to fit the problem. Some that have actually been run include such features as:

-1 Terminal defenses

-2 Mix of area and terminal defenses

-3 Overlapping area defenses

-4 Sensitivity to errors in force estimation

-5 Decoy weapons

-6 Decoy targets

-7 Weapon allocations in sets of 2 or more

-8 Area targets

-9 Area-mobile targets

-10 Targets of variable value

-11 Time-phased attacks

-12 Defense-suppression attacks

-13 Transportation cost matrices

-14 Simultaneous variation of several resource types

-15 A "floating box" to control testing.

In this section, we will discuss general methods of adapting the programs to various types of problems.

Preceding page blank

A. Choice of Program

Generally, if the problem is a one-sided optimization it can best be handled by modifying PATH87A. Examples are terminal-defense problems and transportation problems. Two-sided games require PATH87.

B. Choice of Variables

Resource types, object types, and measures of value must be chosen in an appropriate fashion, keeping the dimensions of the problem as small as possible. In many cases, the choice is obvious; in others, some judgment must be exercised.

In the overlapping area-defense case, each set of interceptors with the same coverage may be considered a resource type, so that there will be as many resource types as there are interceptor sets. Each group of targets covered by the same interceptor sets may then be considered as an object type (but only if these targets are otherwise identical). An incidence matrix may be used to specify the match-up of interceptor types and object types.

In another type of game, the object types were groups of bomber bases, the defense resources were bombers to be allocated to bases and the attack resource types were successive waves of a time-phased attack on the bases, with the potential value per base decreasing as surviving bombers were launched during the attack.

In transportation problems, the destinations are considered as object types and the requirement at each destination as the number of objects of that type. The sources are considered as resource types and the availability at each source as a quantity of resources. If preferred, sources and destinations can be reversed in meaning, with the idea of reducing the number of resource types.

C. Value Function

A value function for point targets is built into PATH87 and PATH87A. It must be replaced when it is inappropriate.

For area targets, the square root law has been used as a value function.

For mobile targets, the region of mobility may be considered as a single target with initial value equal to the number of mobile targets in the region. Depending on whether the region is linear, areal, or spatial, a value function can then be developed to compute expected-value surviving for all allowable elementary attack strategies.

In the bomber-basing case, the initial value of a base is zero, and the maximum value surviving is the number of bombers in the elementary defense strategy for that base. The value function takes into account the time-phasing of the elementary attack strategy and the bomber launches.

In the defense-suppression case, the value function incorporates a suboptimization of the split between attack weapons allocated to radars and those allocated to targets.

In the transportation problem, the value function is a precomputed matrix of costs, one element for each source-destination pair. Costs may be in miles, dollars, or whatever measure is to be minimized.

D. Test Controls

"Floating lid" test controls are used in PATH87 and PATH87A. They should be changed whenever better ones can be devised for a problem.

In the transportation problem, there is never more than one resource unit (weapon) allocated to a destination requirement unit (target). Since tests of two or more would be completely meaningless, the test controls can be greatly simplified.

In the mobile target cases actually run, there were only five allowable elementary strategies on targets of each group. Again, a simple test control procedure was used.

In the bomber-basing case, the results of early runs showed certain systematic patterns in the solution strategies; that knowledge was used to change the test controls to save time on subsequent runs.

E. Economy in Modifications

If a special type of problem is going to be run only a few times, it is usually economical to make minimal modifications to one of the basic programs.

On the other hand, if many runs are planned, it may be economical to make more extensive modifications that take advantage of simplifying features of the problem. For example, if each elementary attack strategy is limited to a single weapon type, each one can be described by identifying the weapon type and the quantity, i.e., by using two numbers instead of a complete vector. In this case, the (A) matrix illustrated in Figure 7, Section IV, can have its blocks reduced to two elements instead of A1 + I(27). Going even further, if the quantity is always one, as in the transportation problem, then only the resource identifying number need be stored in (A). Of course, modifications of this type require that references to (A) be modified throughout the program.

F. Potential Improvements and Applications

Many applications of the PATH programs have been demonstrated. As time goes on, we expect more to be found. We also expect that substantial programing improvements will add to the efficiency, flexibility, reliability, and usefulness of the method.

REFERENCES

1. J. D. Matheson, Preferential Strategies (AD 483 249, AR 66-2, Analytic Services Inc., 1966).

2. J. D. Matheson, S. Endriss, D. Christie, and D. Lake. Preferential Strategies with Imperfect Weapons (AD 813 915, AR 67-1, Analytic Services Inc., 1967).

3. D. F. Dianich and K. E. Hennig. The Weapon Allocation Problem: A Computational Comparison of the Marginal Return Algorithm and a Linear Programming Algorithm (Staff Memorandum, Headquarters Air Force Systems Command, 1970).

4. A. Charnes. "Constrained Games and Linear Programming," Proceedings of the National Academy of Sciences U.S.A., Vol. 39 (1953, pp. 639-641).

5. M. L. Balinski and A. W. Tucker. "Duality Theory of Linear Programs: A Constructive Approach with Applications," SIAM Review, Vol. 11 (1969, pp. 347-377).

6. CALL/360: PL/I Subroutine Library (The Service Bureau Corporation, New York, 1969).

Preceding page blank

SUPPLEMENTARY

INFORMATION

AD-A021 224

Errata - See Item 18

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER SDN 75-3	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) MULTIDIMENSIONAL PREFERENTIAL STRATEGIES		5. TYPE OF REPORT & PERIOD COVERED Strategic Division Note
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John D. Matheson		8. CONTRACT OR GRANT NUMBER(s) F44620-76-C-0010
9. PERFORMING ORGANIZATION NAME AND ADDRESS Analytic Services Inc. (ANSER) 5613 Leesburg Pike, Falls Church, VA 22041		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE December 1975
		13. NUMBER OF PAGES 162
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Directorate of Operational Requirements Hq USAF Wash., D.C. 20330		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited. It may be released to the National Technical Information Service for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Errata, August, [illegible]

Page [illegible], line [illegible]: For [illegible] read [illegible].

Page [illegible], line [illegible]: For [illegible] read [illegible]; move entire line to line [illegible].

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Game Theory	Antimissile Defense
Operations Research	Optimization
Linear Programming	Computer Programming
Parametric Linear Programming	Mathematical Analysis
Strategic Warfare	Military Strategy

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents a method of solving weapon allocation games involving many weapon types and many target types. Numerical solutions are obtained by the PATH method, a form of parametric linear programming. Two computer programs are listed and explained, PATH87 for two-sided games and the simpler PATH87A for one-sided optimizations. Both are copiously illustrated by sample runs. Other applications of the programs are discussed in general terms.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (Continued)

The PATH method offers unique advantages of speed and flexibility in solving problems facing defense analysts, and it is hoped that publication of this report through the National Technical Information Service of the Defense Documentation Center will make this method more widely available. Also, the method has features which can be applied to many problems of resource allocation facing nondefense planners.